UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
ESPECIALIDAD: SONIDO E IMAGEN

PROYECTO FIN DE CARRERA

Diseño e implementación del juego Super Pang para plataforma Android

Autor: Pablo Covarrubias Herrera
Tutor: Juan Peralta Donate

CURSO ACADÉMICO 2011 / 2012

Proyecto Fin de Carrera

DISEÑO E IMPLEMENTACIÓN DEL JUEGO SUPER PANG PARA PLATAFORMA ANDROID

Autor

PABLO COVARRUBIAS HERRERA

Tutor

JUAN PERALTA DONATE

La defensa del presente Proyecto Fin de Carrera se realizó el día __ de _____ de _____, siendo evaluada por el siguiente tribunal:

Presidente:

Vocal:

Secretario:

Y habiendo obtenido la siguiente CALIFICACIÓN:

LEGANÉS, a __ de _____ de 20__

Agradecimientos

Esta memoria significa un punto y final a una de las mejores etapas de mi vida y me gustaría agradecer el apoyo recibido a todas aquellas personas que han estado a mi lado, en lo bueno y en lo malo, durante este tiempo.

A mis padres Pablo y Antonia y mi hermana Marta, por apoyarme desde el primer día e intentar ayudarme cuando las cosas se ponían difíciles, cuando las prácticas no salían o la información para los exámenes no se quedaba en mi cabeza. Gracias por enseñarme todo lo que me habéis enseñado, sin vosotros yo no estaría aquí.

A mis amigos, tanto los de la universidad como los de toda la vida, por estar a mi lado animándome y por los ratos que hemos compartido en este tiempo, en especial a Alberto, Virginia, Estela, Efrén, Cesar, Jessica y el trío de talaveranos.

A Juan Peralta, mi tutor, por darme la oportunidad de realizar este proyecto, por su confianza en mí y por su apoyo y consejos en los momentos que me hacían falta.

Por último y muy especialmente a mi pareja Irene, por su apoyo y sus ánimos en los buenos y malos momentos. Gracias por confiar en mí, estar siempre a mí lado y demostrarme que siempre estarás.

Contenido

Índice de figuras

Índice de tablas

RESUMEN

La situación de la telefonía móvil ha experimentado una gran evolución en los últimos años tanto a nivel funcional como comercial. Uno de los puntos fuertes de esta evolución ha sido la aparición de los smartphones con Android. Si nos fijamos en el sistema operativo Android a día de hoy nos damos cuenta que estamos hablando de un sistema operativo en auge ya que durante el segundo cuatrimestre de 2011, Android ha acaparado el 43% de las ventas de smartphones, situándose líder en su sector con unos 36 millones de unidades vendidas, frente al 17% que obtuvo en el mismo periodo de tiempo del año 2010. Además se prevé que en 2015 las ventas Android ronden el 50% de la cuota de mercado.

Ante estas expectativas se ha decidido realizar una aplicación para dicho sistema operativo. Para ello se aprovechará las posibilidades que ofrece Android en el manejo de acelerómetros y pulsaciones de pantalla para la interacción con el usuario, y las facilidades que se ofrece para mostrar interfaces gráficos.

Se ha elegido realizar una adaptación móvil del videojuego arcade “Super Pang”, el cual dominó en los años 90 las máquinas recreativas y se convirtió en un clásico antes de ser adaptado a videoconsolas como Super Nintendo o Play Station. El juego original consta de dos modos de juego diferentes en los cuales, salvando las diferencias, el objetivo es el mismo: acabar con todas las pelotas que aparecen en pantalla a base de disparos.

ABSTRACT

The situation of mobile telephony both functional and commercial level has experienced a great evolution in last years. One of the strengths of this evolution has been Android smartphones usage. If we look the Android operating system on today, we realize we are talking about an operating system booming because in second four-month period of 2011, Android cornered 43% of smartphone sales, locating leader in his sector with 36 million units sold, compared with 17% of smartphone sales in the same period in 2010. Also provides that Android sales in 2015 will be around 50% market share.

In view of these expectations, an application has wanted to be realized for the above mentioned operating system. For it we will take advantage of the possibilities that Android offers in the managing of accelerometers and pulsations of screen for the interaction with the user, and the facilities that it offers to show graphical interfaces.

It has been chosen to realize a mobile adaptation of the arcade video game called "Super Pang", which dominated in the 90s the recreative machines and became a classic before being adapted to consoles like Super Nintendo or Play Station. The original game consists in two different mode games in which, with some differences, the goal is the same: to finish with all the balls that appear on screen based on shots.

GLOSARIO

QVGA	Quarter Video Graphics Array
IDE	Integrated Development Environment
JDK	Java Development Kit
Java SE	Java Standard Edition
JRE	Java Runtime Environment
SDK	Software Development Kit
ADT	Android Development Tools
UML	Unified Modeling Language
XML	Extensive Markup Language

1

Introducción

En este capítulo introductorio se muestra un enfoque general de los propósitos y objetivos de este proyecto. En él se tratan apartados tales como la motivación, los objetivos, las fases, los medios utilizados y los contenidos.

En el primer apartado se define la **Motivación** 1.1 para la realización de este proyecto, el por qué se ha realizado y que aspectos han influido en su elección y no la de otro proyecto.

El segundo apartado muestra los **Objetivos** 1.2 que se pretenden cumplir en este proyecto, lo que se quiere alcanzar y conseguir con ellos.

Para cumplir con los objetivos del proyecto hay que definir unas **Fases** 1.3 que nos indicarán su ciclo de vida.

También se han querido detallar los **Medios** 1.4 utilizados para la elaboración del proyecto tanto a nivel software como a nivel hardware.

Por último, en los **Contenidos** 1.5 se muestra una visión global del proyecto así como su estructura y un breve resumen de cada apartado.

1.1. Motivación

¿Por qué se ha elegido utilizar Android? ¿Por qué se ha decidido hacer un juego y no cualquier otra aplicación? En este apartado se explican las razones por las cuales se ha elegido este proyecto.

Si analizamos las estadísticas de los smartphone en 2010, solo en Estados Unidos, se vendieron 297 millones de smartphones, lo que supone un 19% del total de teléfonos móviles vendidos (1.6 billones de ventas totales). Estas ventas suponen un aumento del 72% respecto a 2009 y las expectativas esperan un mayor aumento [1].

Según la prestigiosa empresa Gartner (líder mundial en investigaciones de tecnologías de la información), actualmente los terminales con Android suponen el 38.5% del total de smartphones y prevé que en apenas 4 años, Android acapare el 49% de cuota de mercado en el sector móvil [2] como muestra la figura 1.1.

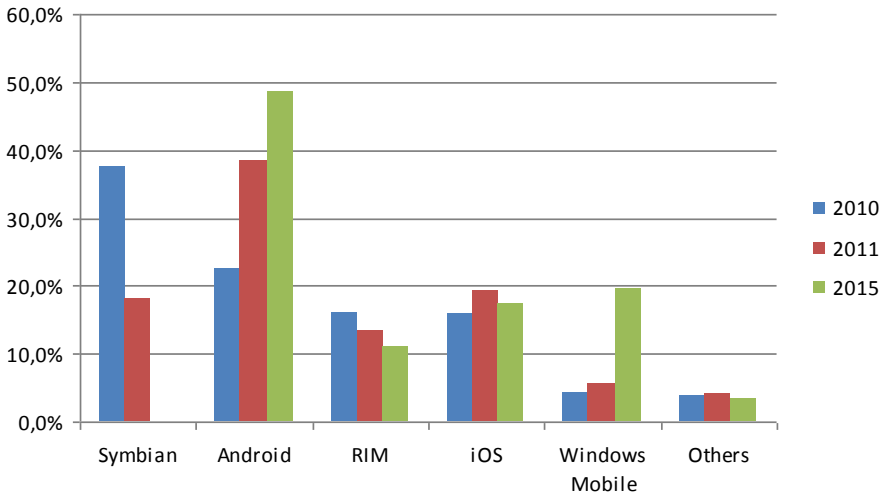


Figura 1.1. Smartphones y sistemas operativos. Estadísticas anuales y previsión.

Centrándose en las aplicaciones en el Android Market, según AndroidZoom, en Septiembre de 2010 había disponibles 169.092 aplicaciones, de las cuales aproximadamente 26.400 estaban destinadas al entretenimiento [3]. Casi un año después, en Agosto de 2011, hay disponibles 277.252 aplicaciones con las que se han conseguido 6 billones de descargas [4].

En la tabla 1.1 se muestran algunos de los juegos más populares, su precio (en euros) y su número de descargas.

Juego	**Nº de descargas**	**Precio**
Angry Birds (Rovio Mobile)	10.000.000 - 50.000.000	Gratis
Live Holden Poker Pro (DragonPlay)	5.000.000 - 10.000.000	Gratis
Texas Poker (KamaGames)	1.000.000 - 5.000.000	Gratis
Doodle Jump (GameHouse)	500.000 - 1.000.000	0.69
Worms (Electronic Arts.)	100.000 - 500.000	1.59
Need For Speed (Electronic Arts)	50.000 - 100.000	2.39
Asphalt 6 HD (Gameloft)	10.000 - 50.000	5.49
Asteroids Defense 2 HD (Deonn Games)	500 - 1.000	2.18

Tabla 1.1. Juegos en Android. Descargas y precios.

A tenor de los datos, se ha considerado interesante la realización de un videojuego para smartphone sobre la plataforma Android, debido a la buena acogida que tiene este tipo de aplicaciones dentro del mercado Android.

El videojuego escogido es una versión para móviles del clásico "Super Pang", conocido en América como Super Buster Bros, que fue desarrollado inicialmente para máquinas recreativas y posteriormente se traslado a las consolas Super Nintendo y Play Station (figura 1.2).

El juego consta de dos modos diferentes y sus objetivos son romper todas las bolas de la pantalla para pasar a la siguiente y explotar todas las pelotas que caen del techo sin parar respectivamente.

Dado que el juego adquirió fama mundial se ha querido diseñar conservando la esencia que le otorgó tanto éxito. La mecánica es tan simple como ir rompiendo bolas pero su dinámica es sumamente adictiva.

Figura 1.2. Super Pang. Versión para Play Station.

1.2. Objetivos

En este apartado se definen los objetivos que se querido tener en cuenta en la elaboración de este proyecto.

El objetivo principal es desarrollar un juego para la plataforma Android, concretamente desarrollar una adaptación móvil de "Super Pang", ofreciendo una alternativa a las consolas y máquinas recreativas.

Otro de los objetivos importantes de este proyecto ha sido tratar de aprovechar las ventajas que ofrece Android en smartphones. Dos de los puntos más importantes de estas ventajas son el uso de la pantalla táctil y los acelerómetros, que conforman la base de la jugabilidad de nuestro juego.

El tercer objetivo que se ha tenido en cuenta ha sido elaborar un proyecto que pueda significar un punto de partida para futuros alumnos, de manera que puedan empezar a crear sus aplicaciones o juegos tomando como base este proyecto.

Para finalizar, el cuarto y último objetivo ha sido crear este proyecto pensando en la posibilidad de futuras expansiones y funcionalidades adicionales de manera que se pueda continuar trabajando con él (Apartado 5).

1.3. Fases del desarrollo

En este apartado se comentan las distintas fases que conforman la elaboración de este proyecto.

- **Documentación previa**: Al elegir este proyecto no se tenían conocimientos sobre Android, de manera que ha sido necesario documentarse sobre esta plataforma y como desarrollar aplicaciones para ella.

- **Análisis de requisitos**: Ha sido necesario evaluar inicialmente los requisitos necesarios para la elaboración de este proyecto y tener en cuenta las funcionalidades que se querían desarrollar.

- **Diseño**: El diseño del proyecto se ha realizado valorando que se quería lograr en cada una de las pantallas y como se quería mostrar al usuario, de manera que sirviera de guía para la implementación posterior.

- **Implementación**: Tras el análisis y diseño del proyecto, se prosigue con la fase de implementación. Al comienzo de dicha fase se consigue tener un juego ejecutable con la funcionalidad más básica, lo que supone que a medida que se profundice en el desarrollo del juego se irá ampliando la funcionalidad del mismo.

1.4. Medios utilizados

Para el desarrollo del presente proyecto se han utilizado elementos hardware y software. Ambos se han utilizado en la fase de desarrollo y pruebas así como en la elaboración de la memoria.

Elementos hardware:

- Ordenador: Se ha utilizado tanto para el desarrollo del juego como para la elaboración de la memoria. No ha sido necesario un ordenador muy potente pues-

to que no se ha utilizado emulador sino que se ha probado directamente sobre un terminal móvil. El modelo utilizado ha sido un Sony Vaio VGN-NS 11S con 4Gb de RAM y sistema operativo Windows Vista.

- Teléfono smartphone: Se ha utilizado para las pruebas durante el desarrollo del juego. Se ha preferido su uso frente al emulador principalmente por la ausencia de acelerómetros en dicho emulador además de ser un elemento más fiable y preciso. En nuestro caso se ha utilizado un HTC Wildfire con pantalla táctil y resolución 320 x 240 QVGA.

Elementos software:

- Eclipse: Versión Helios del año 2010. Con la versión IDE para desarrolladores Java es suficiente.

- JDK: Necesario Java SE, no es suficiente solamente con JRE. Recomendados JDK 5 o JDK 6.

- Android SDK: Imprescindible para programar en Android. La versión utilizada ha sido android-sdk_r05-windows, aunque no se trata de un entorno de desarrollo completo. Solo incluye el núcleo del SDK Tools con el que se puede descargar a continuación el resto de componentes del SDK.

- ADT Plugin para Eclipse: Se trata de un plugin diseñado por Android para el IDE de Eclipse que incluye un set completo de desarrollo y herramientas de depuración. Entre sus funciones se encuentran configurar rápidamente nuevos proyectos, crear interfaces gráficos e incluso exportar aplicaciones firmadas para su distribución.

- Microsoft Office 2003: Se han utilizado herramientas como Microsoft Proyect para la elaboración de los diagramas de Gantt y Microsoft Word y Excel para la elaboración de la presente memoria y gráficas de la misma.

- Gimp 2: Utilizado para el retoque de imágenes.

- Visual Paradigm for UML 8.3: Para la creación de los diferentes diagramas.

1.5. Contenidos

La presente memoria se ha diseñado en varios capítulos en los cuales se profundiza más detalladamente sobre los diferentes aspectos que componen este proyecto.

En el primer capítulo, ***introducción***, se muestra la motivación para la elección de este proyecto, los objetivos que se ha buscado perseguir, las diferentes fases del proyecto y los medios que se han requerido para la elaboración del mismo.

El segundo capítulo, ***estado del arte***, realiza un paso por la historia de los smartphones y se comentan y comparan los sistemas operativos actuales para dichos dispositivos. Se incluye también un repaso de los juegos más populares para Android, se detalla en qué consiste Android y se muestran las diferentes versiones del juego Super Pang.

El tercer capítulo, ***análisis, diseño e implementación***, se trata del capítulo más importante de esta memoria y se puede considerar el núcleo central del proyecto. En él se analiza la estructura del juego, se realiza el diseño y se explica su posterior implementación.

El cuarto capítulo, ***conclusiones***, se analiza el resultado final del proyecto y se medita sobre la consecución o no de los objetivos iniciales.

El quinto capítulo, ***líneas futuras***, se indican las posibles mejoras implementables al proyecto para añadirle funcionalidad o evolucionarlo.

El sexto capítulo, ***fase de testing***, muestra en qué consiste la jugabilidad de un videojuego y las propiedades utilizadas para evaluarlas, así como los resultados del análisis de dichas propiedades y la valoración final de los usuarios.

También se ha querido incluir un capítulo con un ***glosario*** de términos utilizados en la memoria, una ***bibliografía*** y dos ***anexos*** donde se detallan por un lado la planificación y presupuesto del proyecto y por otro se muestran los resultados de las encuestas realizadas a los usuarios.

2

Estado del arte

En este capítulo se ha querido explicar con más detalle los aspectos relacionados con Android y los videojuegos. Relacionados con el primer caso se tratan aspectos como la historia de los smartphones, los diferentes sistemas operativos que pueden utilizar y la historia y evolución de Android. Respecto al uso de juegos, se tratan algunos de los juegos más populares para Android y la historia del juego que nos ocupa, Super Pang.

2.1. Historia de los smartphones

Un smartphone es un término destinado a dispositivos móviles que aúnan funcionalidades de teléfono móvil (realizar o recibir llamadas y mensajes principalmente) con funcionalidades de una PDA (acceder al correo electrónico, organizador personal y actualmente incluso la instalación de diferente software) controladas por un sistema operativo que además gestiona el resto de recursos software y hardware.

La aparición del concepto smartphone, aunque se pueda considerar un término reciente, data de 1993. Debido a su alto coste, los primeros modelos apenas llegaron al público en general y solamente los altos ejecutivos se lo podían permitir [5].

El primer modelo de smartphone fue creado por IBM y recibió el nombre de Simon (figura 2.1). Se trató del primer dispositivo móvil en incorporar servicios de voz y datos (incluía calendario, bloc de notas, correo electrónico, juegos e incluso fax). Utilizaba una pantalla táctil con un teclado QWERTY, tal cual se usa en muchos móviles de la actualidad, pero su peso, tamaño y sobretodo su precio (900 dólares) frenaron su éxito. Su SO se llamaba Zauruz.

Figura 2.1. Simon. El primer smartphone.

En 1997 aparece un prototipo de Ericsson, el modelo GS 88, que nunca se llegó a comercializar y fue el primero en ser calificado de smartphone a pesar de que no llegó al mercado. Un año después apareció el Nokia 9110 Communicator (figura 2.2), tratándose del primer producto de las series communicator de Nokia. Su pantalla aun no admitía colores y no se podía navegar por Internet. Como novedad poseía un teclado QWERTY deslizable que sirvió de inspiración a modelos más actuales. Su sistema operativo era GEOS.

Figura 2.2. Nokia 9110 Communicator.

En 1999, RIM (Research in Motion) sacó a la venta la primera BlackBerry. Se trataba del modelo 850 (figura 2.3) que poseía un teclado físico completo. Revolucionó la forma de comunicación a través de correo electrónico al hacerlo inalámbricamente. Permitía enviar mensajes, enviar y recibir correos electrónicos y actuar como un organizador básico. En su contra hay que decir que tenía una pantalla muy pequeña que tan solo mostraba 8 líneas.

Tres años después RIM volvió a revolucionar el mercado sacando a la venta el modelo 5810 con una pantalla más grande y similar a una PDA. Su principal novedad era la posibilidad de navegar por internet. Ambos modelos montaban el sistema operativo propietario BlackBerry OS y tenían un defecto, necesitaban auriculares para hablar por teléfono ya que no tenían altavoces. Esto se solucionó en 2004 con el modelo 6210.

Figura 2.3. BlackBerry 850. Primer modelo BlackBerry.

En 2003 apareció el modelo Palm Treo 600 (figura 2.4), que poseía teclado QWERTY retro-iluminado, cámara integrada y pantalla a color. Otra importante novedad era el soporte de redes GSM y CDMA. Contaba con 32 Mb de RAM y un procesador de 144 MHz Este modelo dominó el mercado estadounidense de smartphones durante varios años y también montaba un sistema operativo propietario, PalmOS.

Figura 2.4. Palm Treo 600.

En 2007 se producen dos acontecimientos importantes en el mercado smartphone. Por un lado Apple lanzo su primer iPhone (figura 2.5), también con sistema operativo propietario, iOS, el cual tuvo un gran éxito gracias a su novedosa pantalla táctil y a una gran experiencia de navegación por Internet. Poseía una cámara de 2 megapíxeles e in-

cluía conectividad WiFi. Supuso un punto de inflexión en la concepción de los teléfonos smartphones

Figura 2.5. iPhone. Primer modelo.

Por el otro Google liberó la primera versión de Android para desarrolladores. Inicialmente no causó un gran revuelo pero hoy en día se trata del SO para móviles con mejor previsión de futuro. Este dato cobra más relevancia si cabe si se tiene en cuenta la cantidad de SO de calidad que existen actualmente (Symbian, Windows Phone, iOS,..).

En 2008 Apple lanzo la evolución del iPhone, el modelo 3G, el cual incluía la posibilidad de acceder a redes 3G y tan solo un año después lanzó el modelo 3GS.

En este mismo año, 2009, Palm anuncia WebOS (la evolución de PalmOS) y apareció en escena el Motorola Droid (figura 2.6), el cual supuso el primer gran éxito del SO Android en EEUU, vendiéndose 1 millón de unidades en apenas 74 días. Su gran novedad es el uso de la versión 2.0 del sistema operativo.

Figura 2.6. Motorola Droid. Primer gran éxito de Android.

En 2010 apareció el HTC EVO 4G (figura 2.7), un móvil que portaba Android y que logra aprovechar al máximo el potencial de la red WiMax (la red inalámbrica más rápida en EEUU). Además es destacable su amplia pantalla de 4.3" con una resolución 800x400.

Figura 2.7. HTC EVO 4G.

Por último, 2011 se ha convertido en uno de los años con mayores novedades. Las principales novedades son el aprovechamiento de las redes 4G, las pantallas en alta resolución y el gran aumento de la potencia de los procesadores, que comienzan a permitir la visualización de videos y juegos en HD. Algunos de estos nuevos terminales son los módelos iPhone 4, Google Nexus S o Samsung Galaxy S.

2.2. Sistemas operativos para smartphones

Un SO móvil es un sistema operativo que controla un dispositivo móvil análogamente a como Windows, Linux o MAC controlan un PC, es decir, controlan tanto los recursos hardware como software, con la diferencia que estos SO móviles son más simples y están más orientados a la conectividad inalámbrica y los formatos multimedia.

En la actualidad el mercado smartphone se lo reparten 6 compañías con sus respectivos sistemas operativos. Estos sistemas operativos son Android, iOS, BlackBerry OS, Microsoft Windows Phone, Symbian, y Bada.

- **Android**: Se trata de un SO basado en Linux que se ejecuta en Java, desarrollado por Google. En 2007 se lanzó el primer kit para desarrolladores y desde entonces se ha ido actualizando el sistema estando disponible incluso para dispositivos como tablets o netbooks.

- **iOS**: Sistema operativo desarrollado por Apple. Inicialmente se diseñó para el dispositivo iPhone pero su uso se ha extendido a otros dispositivos propiedad de Apple como iPod e iPad.

- **BlackBerry OS**: Desarrollado por Research In Motion (RIM) para sus dispositivos BlackBerry, está basado en Java y emplea MIDP sobre CLDC. Soporta multitarea y diferentes métodos de entrada como trackwheel, trackball, touchpad y pantallas táctiles.

- **Microsoft Windows Phone**: Sistema operativo propietario de Microsoft. Soporta pantallas táctiles de alta resolución, integración con redes sociales y multiescritorio. Por el contrario se olvida la sincronización de datos con el PC, transferencia por Bluetooth e intercambio de tarjetas de memoria externas.

- **Symbian OS**: Desarrollado por Symbian Ltd. que fue fundada por la unión de varias empresas de telefonía móvil como Nokia, Motorola o Ericsson entre otras. Permite un uso eficiente de memoria y energía del dispositivo y soporta en tiempo real los protocolos de comunicación y telefonía. Además soporta múltiples lenguajes de programación como Symbian C++, Java ME, Open C, etc. Su futuro es incierto ya que actualmente es propiedad de Nokia, quien en febrero de 2011 firmó una alianza con Microsoft para usar Windows Phone.

- **Bada**: Sistema operativo desarrollado por Samsung que permite cualquiera de los kernel de Linux. Sus aplicaciones son desarrolladas en C++ y ofrece varios controles de interfaz de usuario a los desarrolladores. También soporta diversos sensores, GPS y detección de rostros. No permite instalar aplicaciones fuera de la tienda ni usar programas tipo VoIP/SIP [6].

- **Otros**: Otros SO que a día de hoy no forman parte de las estadísticas de sistemas operativos más relevantes son WebOS, Maemo/MeeGo o LiMo, aunque estos dos últimos sistemas recientemente se han unido dando lugar a un sistema operativo llamado Tizen [7].

Según el análisis de Gartner del segundo cuatrimestre del año 2011 [8], la venta de dispositivos móviles se ha incrementado un 16.5% mientras que la venta de smartphones ha aumentado un 74% respecto al mismo periodo de 2010, constituyendo el 25% de las ventas totales.

Dicho análisis (tabla 2.1) también muestra el aumento de ventas de teléfonos con Android, iOS o Bada, y un decremento de ventas de Symbian, Windows Phone y RIM. El principal motivo de caída de Symbian se encuentra en la disminución de ventas de Nokia, principal portador de dicho sistema operativo.

Operating System	2Q11 Units	2Q11 Market Share (%)	2Q10 Units	2Q10 Market Share (%)
Android	46,775.9	43.4	10,652.7	17.2
Symbian	23,853.2	22.1	23,386.8	40.9
iOS	19,622.8	18.2	8,743.0	14.1
BlackBerry OS	12,652.3	11.7	11,628.8	18.7
Bada	2,055.8	1.9	577.0	0.9
Microsoft Windows Phone	1,723.8	1.6	3,058.8	4.9
Others	1,050.6	1.0	2,010.9	3.2
Total	**107,704.4**	**100.0**	**62,058.1**	**100.0**

Tabla 2.1. Sistemas operativos en smartphone. Ventas en el segundo cuatrimestre de 2010 y 2011.

Analizando las ventas a usuarios finales [9], Gartner nos muestra un incremento casi exponencial del sistema operativo Android alcanzando unas ventas de 46.8 millones de unidades, seguido de Symbian con 23.9 millones y de iOS con 19.6 millones (figura 2.8).

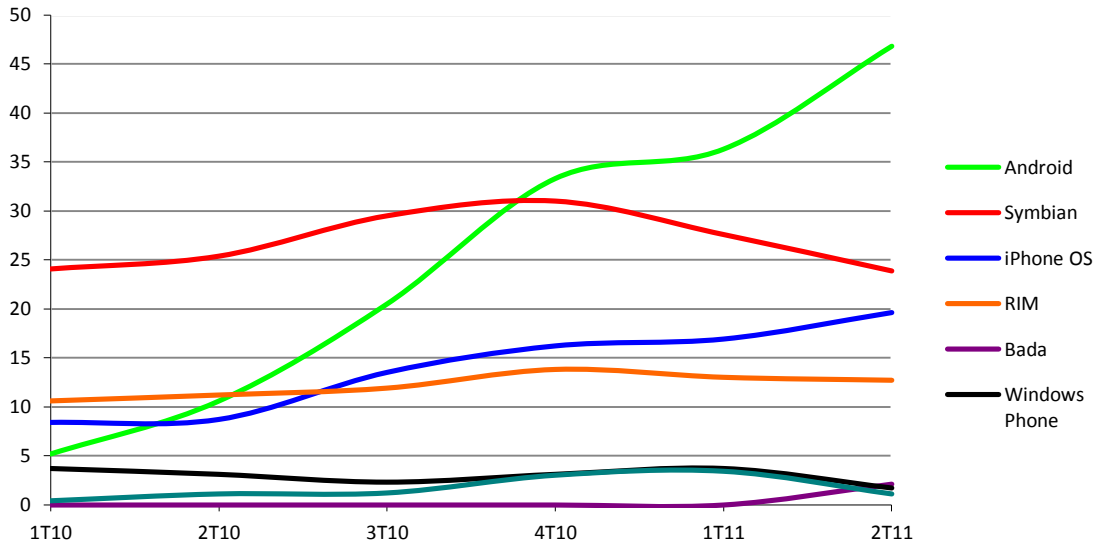


Figura 2.8. Sistemas operativos para smartphone. Evolución trimestral 2010-11.

Técnicamente no existe el sistema operativo ideal, pero comparándolos entre sí se pueden ver las ventajas de unos y otros y se puede evaluar para cada caso que sistema operativo puede resultar más adecuado [10].

Dos de los parámetros más importantes en la elección del sistema operativo son el interfaz de usuario y el conjunto de aplicaciones que tenga disponibles.

Si comparamos el interfaz de usuario de los distintos sistemas operativos que se han analizado tanto iOS como Android y Windows Phone 7 ofrecen un apartado visual bonito, fluido y con buena respuesta, aunque de los tres sistemas Android es el único que ofrece una personalización total por ser un sistema operativo gratuito.

Si evaluamos la cantidad de aplicaciones, tanto por calidad como por cantidad que hay detrás de cada sistema operativo, el mejor sistema seria iOS seguido cada vez más de cerca por Android y un poco más lejos por Symbian.

Otro factor a tener en cuenta es el rendimiento del dispositivo. En este caso Symbian es claramente el sistema que más eficientemente gestiona la energía y batería del dispositivo, aunque Windows Phone parece ser que también funciona bien, mientras que éste es el punto débil de Android e iOS.

Por último se debe analizar la funcionalidad de los sistemas. Si bien inicialmente tanto iPhone como Windows Mobile arrancaron bastante escasos de funcionalidad, con sus nuevas actualizaciones iOS 4.3 y Windows Phone 7 parecen estar bastante completos aunque éste último no soporta aun multitarea ni Flash (iOS tampoco soporta Flash). En éste aspecto Android es el más completo ya que soportan multitarea, multitáctil, copiar y pegar, Flash, widgets, WiFi Tethering, etc.

Actualmente y a tenor de los datos analizados, se ha considerado que Android es el sistema operativo que ofrece una mejor funcionalidad y experiencia de usuario además de ofrecer un buen soporte a los desarrolladores ya que es software libre, lo cual se ha tenido en cuenta a la hora de desarrollar el juego.

2.3. Aplicaciones y juegos para smartphones

Hoy en día las aplicaciones son un aspecto básico de un smartphone y posiblemente un sistema operativo que no posea un buen catálogo de aplicaciones se quede fuera de la lucha por el mercado móvil. La naturaleza de las aplicaciones es muy variada, desde lectores de periódicos hasta juegos pasando por aplicaciones GPS o redes sociales por poner solo un ejemplo.

En los diferentes stores de cada plataforma, el número de aplicaciones disponible aumenta cada día que pasa de manera que se convierten en un punto muy importante de cada sistema operativo. Este crecimiento demuestra que los smartphone son los dispositivos llamados a suceder al teléfono convencional.

2.3.1. Stores de aplicaciones

Actualmente cada uno de los sistemas operativos destinados a ser utilizados en un smartphone posee su propio store de aplicaciones a los cuales se puede acceder a través del propio móvil y en algunos casos incluso desde el PC.

- Android Market: Store del sistema operativo Android. Su lanzamiento se produjo en octubre de 2008 y tres años después el número de aplicaciones activa es 319.161.

- App Store: Store de Apple. Su lanzamiento se produjo en julio de 2008 y en octubre de 2011 el número de aplicaciones activas es algo mayor al de Android Market: 459.589.

- Marketplace: Store de Windows Phone y su lanzamiento se produjo en octubre de 2009. Dos años más tarde, en octubre de 2011, el número de aplicaciones es 35.000.

- App World: Se trata del store de BlackBerry OS. Su salida al mercado se produjo en abril de 2009 y en julio de 2011 disponía de más de 40.000 aplicaciones.

- Nokia Store: Store de Symbian OS que inicialmente se llamó Nokia Ovi Store. Nació en mayo de 2009 y en abril de 2011 ya tenía 50.000 aplicaciones

- Palm App Catalog: Store de WebOS que se lanzó en junio de 2009 con apenas 18 aplicaciones y en septiembre de 2010 ya estaban disponibles 4000 aplicaciones oficiales.

2.3.2. Tipos de aplicaciones

Actualmente hay muchos tipos de aplicaciones (negocios, bolsa, meteorológicas, deportivas, etc.) pero centrándose en el market de Android se pueden clasificar en dos grandes grupos: aplicaciones y juegos.

- Juegos: Hay gran variedad de tipos: arcade, carreras, deportes, cartas, puzzle e incluso widgets están incluidos en esta categoría.

- Aplicaciones: Cualquier aplicación que no sea juego ni widget, es decir, aplicaciones relacionadas con compras, redes sociales, finanzas, medicina, etc.

Por lo general, tanto para el market de Android como para cualquier otra tienda de aplicaciones de otro sistema operativo, las aplicaciones que más éxito suelen tener son aquellas gratuitas.

2.3.3 Juegos más populares para Android

En este apartado se comentan los juegos más populares que posee Android. Muchos de ellos sirven como inspiración para futuros juegos e incluso han marcado tendencia. No todos son exclusivos de Android, algunos como Angry Birds son originales de iOS y han sido adaptados a este sistema operativo.

Evaluando diferentes listas de internet sobre los juegos gratuitos más populares se ha elaborado otra con aquellos que han tenido más éxito:

1. Angry Birds: El más popular en la era smartphone. En este juego se lanza con un tirachinas unos pájaros muy enfadados que quieren acabar con unos cerdos que les roban sus huevos. Tiene más de 10 millones de descargas (figura 2.9).

2. PaperToss: Tan simple como encestar una bola de papel en una papelera pero adictivo como pocos. Ha sido descargado más de 10 millones de veces.

3. Jewels: Clásico juego en el que hay que juntar 3 figuras iguales para sumar puntos. Este simple pero muy divertido juego ha sido descargado más de 10 millones de veces.

4. Air Control Lite: En este juego deberemos controlar el tráfico aéreo y evitar colisiones entre aviones. También tiene más de 10 millones de descargas.

5. Drag Racing: Los buenos juegos de carreras de coches han sido siempre bien valorados y este juego ya ha sido descargado más de 10 millones de veces.

6. Robo Defense: Clásico juego del estilo “defiende la torre” en el que aparecerán enemigos y tendremos que impedir que entren en la fortaleza. Descargado más de 5 millones de veces (figura 2.10).

7. World War: Juego que solo posee interfaz gráfica y ofrece interacción con el usuario a través de botones, en el que debemos cumplir misiones y atacar a otros jugadores para ganar dinero y aumentar nuestro ejército. Descargado más de 5 millones de veces.

8. Live Holden Poker Pro: Este juego multijugador en el que hay que acumular dinero jugando al póker ha sido descargado más de 5 millones de veces.

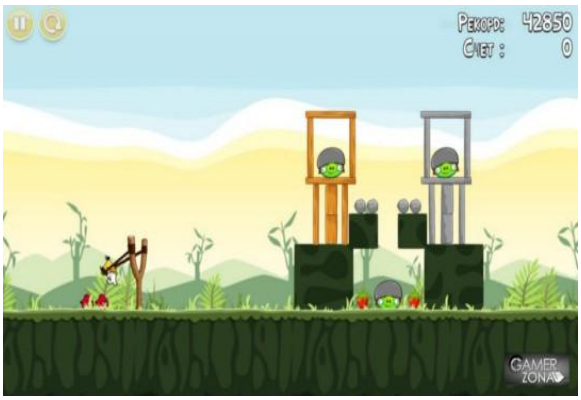


Figura 2.9. Angry Birds. Versión para Android.

Figura 2.10. Robo Defense. Juego para Android.

2.4. Historia de Android

En 2003 se creó la empresa Android Inc. en cuyos inicios se gestó la idea de desarrollar un sistema operativo para móviles, al que llamaron Android. Esta compañía fue comprada por Google en 2005 [11].

Dos años después se anunció Android como un sistema abierto basado en GNU/Linux y orientado al mercado de los llamados teléfonos inteligentes. Fue el ingreso de Google en el mercado de la telefonía móvil.

En 2007 se anuncio la creación de la Open Handset Alliance [12], una alianza de varias compañías dedicadas al sector de los dispositivos móviles. En dicha presentación se anunció que Android sería su apuesta como sistema operativo y su intención era que

fuera abierto y gratuito gracias, entre otras cosas, a su kernel basado en un kernel Linux, lo que supondría la posibilidad de adaptarlo a casi cualquier terminal móvil.

A los pocos días de la presentación del sistema operativo se anuncio la disponibilidad del SDK de Android junto a un emulador, orientado a los futuros desarrolladores de aplicaciones Android [13].

El primer teléfono móvil que cargaba Android fue el HTC G1 (o HTC Dream) lanzado en octubre de 2009 con la versión 1.1 y desde ese momento multitud de fabricantes (LG, HTC, Sony Ericcson, etc.) han incorporado Android a sus dispositivos móviles en sus diferentes versiones.

Progresivamente y a medida que Android ha ido adaptándose a diferentes operadoras y compañías, se ha podido comprobar un concepto conocido como fragmentación, que supone la aparición de diferentes versiones y la actualización aleatoria por parte de las diferentes compañías y operadoras de sus terminales móviles.

El primer gran terminal que salió a la venta portando Android fue el primer móvil de Google y fabricado por HTC, llamado Nexus One, que además llevaba incorporada la nueva actualización del sistema operativo, la versión 2.1.

En mayo de 2010, Android dio el gran salto con la actualización 2.2 que trajo bastantes mejoras: Adobe Flash Player, optimización de WiFi, Bluetooth, mejora del rendimiento de batería, etc. y aparecieron grandes modelos de terminal como el HTC Desire o Samsung Galaxy S. A finales de año apareció la última versión disponible para smartphones, la 2.3, que aun no ha sido aprovechada por la mayoría de terminales y ofrece novedades como soporte nativo para VoIP, SIP o soporte NFC.

Por último, Android no es solo un sistema operativo para smartphones, también ha sido incorporado a las tablets PC y netbooks con una versión exclusiva, la 3.0.

2.4.1. ¿Qué es Android?

Se trata de un conjunto de software para dispositivos móviles formado por sistema operativo, middleware y aplicaciones [14].

Entre sus características se encuentran un marco de aplicaciones que permite reusar y reemplazar componentes, una maquina virtual Dalvik optimizada para dispositivos móviles, un navegador integrado basado en el motor WebKit, gráficos optimizados 2D y 3D basados en OpenGL o SQLite para almacenamiento de datos estructurados entre otras características.

El SDK de Android proporciona las herramientas y APIs necesarias para comenzar a desarrollar aplicaciones utilizando Java como lenguaje de programación.

2.4.2. Arquitectura de Android

La imagen a continuación (figura 2.11) muestra como está estructurado el sistema operativo y los componentes de cada sección.

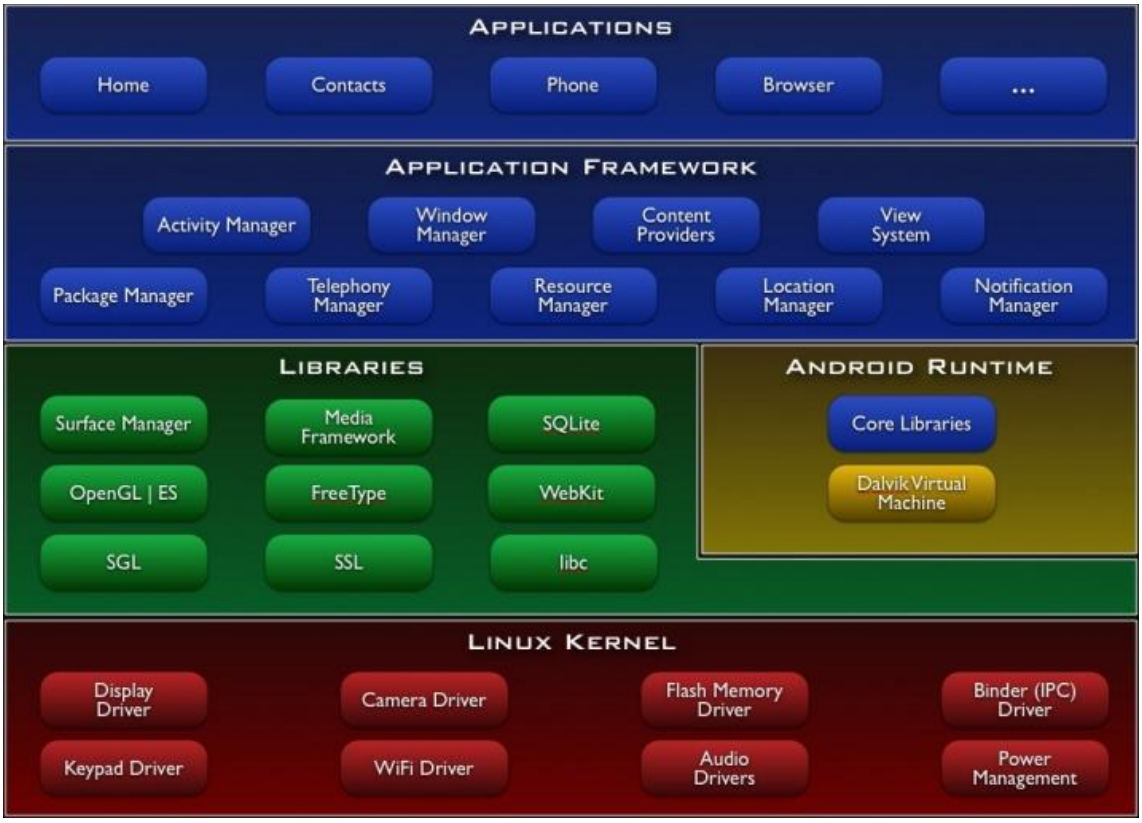


Figura 2.11. Arquitectura del sistema operativo Android.

A continuación se explican más detalladamente cada una de las secciones de la arquitectura.

Aplicaciones:

Se trata de un conjunto de aplicaciones básicas escritas en Java como e-mail, calendario, aplicación de SMS, mapas, navegador, etc. que suelen venir pre-instaladas en el dispositivo móvil.

Framework de aplicaciones:

Compuesto por un conjunto de herramientas para el desarrollo de aplicaciones. Al ofrecer una plataforma de desarrollo abierta, Android ofrece a los desarrolladores la capacidad de crear aplicaciones innovadoras y extremadamente ricas visualmente, además de poderse beneficiar del hardware del controlador, acceder a información de localización, correr servicios en background, etc.

Los desarrolladores podrán acceder a las mismas APIs usadas por las aplicaciones básicas. La arquitectura de la aplicación está diseñada para reutilizar componentes, es decir, cualquier aplicación puede publicar sus capacidades y cualquier otra puede hacer uso de las mismas.

Por debajo de todas las aplicaciones se encuentran un conjunto de servicios y sistemas, incluyendo:

- Un amplio conjunto de vistas (*Views*) que se pueden usar para construir una aplicación como listas, grids, cajas de texto, botones, etc. e incluso un navegador embebido.

- *Content Providers* que permiten a las aplicaciones acceder a datos de otras aplicaciones o compartir sus propios datos.

- *Resource Manager* para poder acceder a recursos como Strings, gráficos y ficheros XML.

- *Notification Manager* que permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.

- *Activity Manager* para manejar el ciclo de vida de las aplicaciones y ofrecer una navegación común.

Librerías

Compuesto por un conjunto de librerías escritas en C/C++ usadas por varios componentes del sistema, cuyas capacidades son puestas a disposición del desarrollador a través del framework de aplicaciones.

Algunas de las librerías principales son: System C, librerías Media, Surface Manager, LibWebCore, SGL, librerías 3D, FreeType y SQLite.

Android Runtime

Compuesto por las librerías principales y la máquina virtual Dalvik. Las librerías principales permiten que la funcionalidad disponible en las librerías anteriores esté disponible desde las librerías principales de Java. La maquina virtual Dalvik se ha diseñado de forma que cada aplicación pueda correr en su propio proceso con su propia instancia de la maquina virtual, de manera que un dispositivo móvil pueda correr múltiples maquinas virtuales simultáneamente que ejecutan ficheros en el formato .dex optimizado para minimizar el uso de memoria.

Linux Kernel

Android se basa en una versión de Linux 2.6 para servicios del sistema principales como seguridad, manejo de memoria, manejo de procesos, pila de red o modelo de controladores. Además actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

2.4.3. Fundamentos de las aplicaciones

Las aplicaciones Android están escritas en lenguaje de programación Java [15]. Las herramientas del SDK compilan tanto el código como los ficheros de datos y otros recursos en un único fichero con extensión .apk que incluye todo el código de la aplicación y es el fichero que utiliza el dispositivo para instalarlas.

Cuando la aplicación ha sido instalada en el dispositivo móvil, vive en su propia zona de seguridad. Esto se consigue de la siguiente forma:

- Android es un sistema Linux multiusuario y cada aplicación se considera un usuario diferente.

- Android asigna un ID de usuario único a cada aplicación (este identificador es conocido por el sistema y no por la aplicación). El sistema establece unos permisos para todos los ficheros de una aplicación de manera que solo el ID de usuario asignado a esa aplicación puede acceder a ellos.

- Cada proceso corre en su propia maquina virtual de manera que está aislada del resto de aplicaciones.

- Cada aplicación corre en su propio proceso Linux que Android arranca cuando cualquier componente de la aplicación necesita ser ejecutado y que Android cierra cuando ya no se necesita o es necesario recuperar memoria para otras aplicaciones.

Android utiliza el principio de privilegios mínimos, haciendo que una aplicación solo pueda acceder a los componentes que requiere para funcionar y ninguno más, de manera que no puede acceder a partes del sistema a las que no tiene permisos.

Sin embargo es posible que varias aplicaciones compartan datos si comparten el mismo ID de usuario, permitiéndolas incluso compartir la misma máquina virtual y el mismo proceso. También es posible que una aplicación acceda a los servicios del sistema, como almacenamiento en la tarjeta SD, contactos de teléfono, etc. Para ello el usuario debe conceder a la aplicación ciertos permisos a la hora de instalarla en el dispositivo.

Los componentes de las aplicaciones son los bloques esenciales para la construcción de una aplicación. Cada componente es un punto diferente a través del cual el sistema es capaz de acceder a tu aplicación, aunque no todos los componentes son puntos de entrada reales para el usuario e incluso pueden depender de otros componentes. A pesar de ello, cada componente tiene entidad propia y juega un rol específico en el comportamiento de la aplicación.

Existen cuatro tipos de componentes, cada uno con un propósito y ciclo de vida distinto:

- **Actividades**: Representa una pantalla única con interfaz de usuario. Una actividad es implementada como subclase de Activity y es independiente del resto de actividades de una aplicación. El ciclo de vida de una actividad se muestra en la figura 2.12.

- **Servicios**: Representa una tarea sin interfaz gráfico que se ejecuta de fondo. Otro componente puede arrancarlo, interactuar con él, dejarlo correr o pararlo. Un servicio es implementado como una subclase de Service.

- **Proveedor de contenidos**: Se encarga de manejar datos compartidos entre aplicaciones, almacenados en cualquier localización de almacenamiento persistente, que pueden ser consultados o modificados. Se implementa como una subclase de ContentProvider.

- **Receptores broadcast**: Son componentes que responden a los anuncios broadcast originados por el sistema u otras aplicaciones. No tiene interfaz gráfico pero puede enviar notificaciones a la barra de estado del sistema. Se implementan como una subclase de BroadcastReceiver.

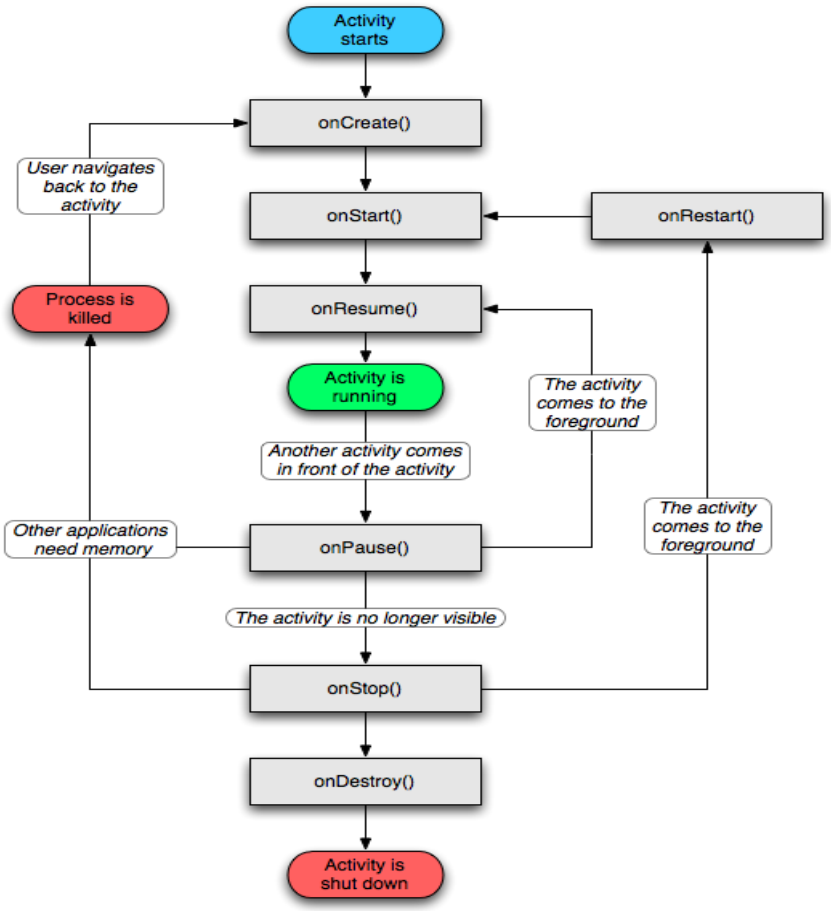


Figura 2.12. Ciclo de vida de una actividad.

Un aspecto único de Android es que una aplicación puede arrancar cualquier componente de otra aplicación sin necesidad de implementar el código, basta con acceder a la actividad de la aplicación que pueda realizar la tarea que necesitas.

Estos componentes, excepto ContentProvider, se activan mediante el uso de un mensaje asíncrono llamado Intent, cuya finalidad es enlazar dos componentes en tiempo real. En el caso de Activity o Service este mensaje representa una acción a realizar.

Antes de poder usar estos componentes es necesario declararlos en un archivo llamado AndroidManifest.xml, en el cual se deben incluir también los permisos que necesita la aplicación, el nivel de API mínimo que se requiere, requerimientos hardware y software necesarios, etc.

2.5. Historia de Super Pang

Super Pang es un videojuego arcade de la década de los 90 que se convirtió en un clásico de las máquinas recreativas. Se trata de la secuela de un juego desarrollado por Mitchell Corporation en 1989 llamado Pang en España, Pomping World en Japón o Super Buster Bros en América, que inicialmente se lanzo sólo para máquinas recreativas [16].

La historia de Pang (figura 2.13) es simple, dos niños vestidos de explorador recorren el mundo amenazados por unos extraños globos de colores con los que al chocar pierden una vida. El juego tiene jugabilidad para dos jugadores y diversidad de armas e ítems. Dispone de 50 niveles que recorrían gran diversidad de paisajes, Japón, Egipto, Kenia e incluso Barcelona estan representados a lo largo de la historia y fue adaptado a diferentes plataformas como PC-Engine, Commodore 64, Atari o Spectrum entre otras.

Figura 2.13. Pang. Versión arcade.

Un año después, en 1990, apareció en Japón la secuela de Pang, llamada Super Pang (figura 2.14). La temática es muy similar aunque gráficamente hay alguna diferencia respecto al Pang original, las ilustraciones de fondo están más cuidadas, nuestros exploradores pasan a ser dos jóvenes con gorra y chaquetas a juego azul y roja, y la jugabilidad resulta algo más fluida.

El modo de juego principal, llamado Tour Mode, mantiene la esencia de su predecesor acabando con todos los globos de colores en distintos paisajes, entre los que se encuentran Hong-Kong, Alemania, Venecia o el Alcázar de Segovia, cuya dificultad irá en aumento a medida que avancemos de pantallas. En esta versión aparece un nuevo modo, llamado Panic Mode, en el que aparecen bolas desde el techo de la pantalla sin parar y a medida que se rompan aumentará una barra de colores que al llenarse aumentará automáticamente un nivel la pantalla. Por supuesto al subir niveles, subirá la dificultad, hasta alcanzar un máximo de 99 niveles.

Aunque su jugabilidad es la misma que la de Pang, trae alguna novedad como por ejemplo bolas de nuevos colores, bolas que botan más que las estándar y un tipo de glo-

bo con forma geométrica romboide que se mueve linealmente por la pantalla. Además de los ya comentados globos de colores, aparecen unos seres con los que se interactúa durante la pantalla, que también aparecían en Pang, y podían ayudarte a terminar la pantalla o dejarte imposibilitado para disparar.

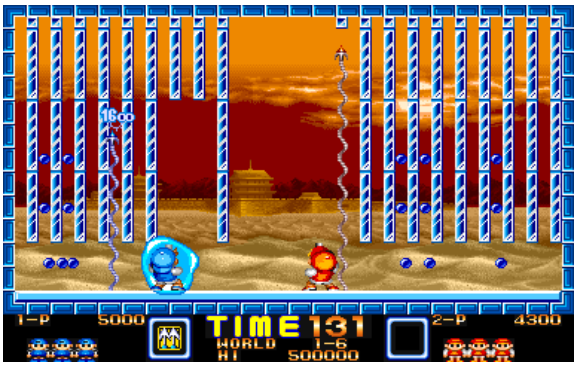


Figura 2.14. Super Pang. Versión arcade.

Este Super Pang fue adaptado y distribuido por la compañía Capcom para las consolas Super Nintendo y Play Station. La versión para la consola de 16 bits apareció en 1992 pero no llegó a España y tan solo ofrecía jugabilidad para un jugador. Capcom introdujo algunas melodías suyas eliminando algunas originales y gráficamente no llega a la altura que llego la versión arcade. La versión en 32 bits llegó en un formato pack junto a su antecesor, Pang, y su sucesor, Pang! 3, ofreciendo un mejor nivel que la versión para Super Nintendo, siendo bastante más fiel al original.

Dicho sucesor, Pang! 3, apareció en el mundo arcade en 1995 y se aleja de la temática exploradora, mostrándonos un juego bastante más paródico (figura 2.15). Esta vez nos muestra obras de arte históricas en lugar de paisajes mundiales. Añade un modo de juego, Normal Mode, y un modo llamado Beginner que servía como tutorial para mostrar las novedades del juego.

Figura 2.15. Pang! 3. Versión para PlayStation.

3

Análisis, diseño e implementación

En este capítulo se detalla el proceso de creación del Proyecto Fin de Carrera, desde la primera idea hasta la instalación de la aplicación en un dispositivo móvil. Se puede considerar el capítulo más importante ya que se explican en profundidad los pasos que se han seguido para desarrollar esta aplicación.

Se ha dividido en tres partes para explicar cada una de ellas en detalle por separado. En la primera se muestra un análisis inicial del proyecto y sus primeros pasos. La segunda parte se centra en el diseño realizado y las características del juego. En la tercera y última parte se detalla la implementación del juego.

3.1. Análisis

3.1.1. Propuesta inicial

La idea planteada inicialmente fue la realización de un juego sobre la plataforma Android sin especificar tipo ni juego concreto. La elección del juego fue personal y se decidió implementar el juego Super Pang, un juego en dos dimensiones en el que un jugador debe destruir todas las bolas de colores de la pantalla.

La pantalla se divide en dos zonas. La zona inferior está destinada simplemente a mostrar información válida para el usuario con datos como el número de vidas restante, la puntuación actual, tiempo restante para finalizar el nivel o el nivel en que se encuentra. La zona superior es donde se desarrolla toda la acción del juego. Todas las pantallas y niveles del juego son predeterminadas y no se deja nada aleatorio.

Al iniciar la aplicación se carga un menú principal (emitiendo un sonido con el nombre del juego) en el que el jugador puede elegir entre iniciar el juego para pasar a

una pantalla de elección de modalidad, elegir entre una serie de opciones o visualizar las instrucciones para saber cómo funcionan los modos de juego.

El jugador debe poder elegir entre dos modos distintos de juego, uno en el que avance por pantallas distintas y otro en el que no haya pausa salvo en caso de victoria o fin de partida. En ambas modalidades el nivel de dificultad se incrementará automáticamente con el avance de pantallas o niveles.

Si el modo elegido es el llamado Panic Mode, el jugador solo debe preocuparse de disparar a las bolas y esquivarlas con el objetivo de aumentar la barra de nivel y conseguir llegar al nivel final, mientras que si elige el llamado Tour Mode, cada pantalla contendrá un número predeterminado de bolas que deberá explotar antes de que termine el tiempo (si se consigue se emite una melodía y se carga la siguiente pantalla). En este segundo modo podrá, además, romper bloques que aparecen en pantalla pudiendo conseguir diferentes items para sumar mayor puntuación, cambiar de disparo o conseguir vidas extra (tanto la ruptura de bloques y bolas como la obtención de los citados items emiten un sonido).

Independientemente del modo de juego que el jugador ha escogido, al llegar a cero su contador de vidas, se emite una melodía indicando el fin de juego y se muestra una pantalla con las máximas puntuaciones. Si la conseguida por el jugador forma parte de las cinco primeras, se le da la posibilidad de introducir un identificativo para su puntuación.

Si el jugador consigue finalizar una pantalla en Tour Mode o el último nivel en Panic Mode, se emitirá una melodía a modo de victoria y se muestra una pantalla con las máximas puntuaciones del modo elegido, permitiendo igualmente introducir un identificativo para su puntuación en caso de encontrarse entre las mejores.

3.1.2. Requisitos de usuario

Tras el análisis de la propuesta inicial, se realiza la evaluación de los requisitos de usuario. En dichos requisitos, el propio usuario indica un conjunto de restricciones o funcionalidades que el programa, juego o aplicación en cuestión debe cumplir. Se diferencian dos tipos de requisitos: de capacidades (requeridos por el usuario para resolver un problema o determinar un objetivo) y de restricciones (impuestos por los usuarios sobre cómo debe ser resuelto el problema o logrado el objetivo).

Los requisitos de usuario están compuestos por los siguientes campos:

- **Identificador**: Muestra el tipo de requisito RUC (requisito de usuario de capacidad) o RUR (requisito de usuario de restricción) acompañado de un valor numérico. El identificador debe ser un valor único.

- **Necesidad**: Muestra la prioridad del requisito. Su valor puede ser Esencial o Deseable.

- **Título**: Frase breve que indica el nombre del requisito.

- **Descripción**: Breve comentario que describe el requisito.

A continuación se muestran los requisitos de usuario de capacidad:

Identificador:	RUC-01	Necesidad:	Esencial
Título:	Arranque del juego		
Descripción:	El juego se arrancará a través de un ejecutable instalado en el propio terminal.		

Tabla 3.1. RUC-01. Arranque del juego.

Identificador:	RUC-02	Necesidad:	Deseable
Titulo:	Pantalla predeterminada		
Descripción:	Los niveles del modo Tour deben generarse de forma predeterminada.		

Tabla 3.2. RUC-02. Pantalla predeterminada.

Identificador:	RUC-03	Necesidad:	Deseable
Título:	Pantalla aleatoria		
Descripción:	Los niveles del modo Panic se deben generar de forma aleatoria (las bolas aparecen en posiciones aleatorias)		

Tabla 3.3. RUC-03. Pantalla aleatoria.

Identificador:	RUC-04	Necesidad:	Esencial
Titulo:	Instrucciones del juego		
Descripción:	Se deben poder mostrar las instrucciones del juego		

Tabla 3.4. RUC-04. Instrucciones del juego.

Identificador:	RUC-05	Necesidad:	Deseable
Titulo:	Opciones del juego		
Descripción:	El usuario debe poder modificar las diferentes opciones del juego.		

Tabla 3.5. RUC-05. Opciones del juego.

Identificador:	RUC-06	Necesidad:	Deseable
Título:	Mensaje de pausa		
Descripción:	Se debe mostrar un mensaje cuando se pausa el juego, permitiendo al jugador seguir jugando o abandonar la partida.		

Tabla 3.6. RUC-06. Mensaje de pausa.

Identificador:	RUC-07	Necesidad:	Deseable
Título:	Mensaje para abandonar la aplicación		
Descripción:	En el menú principal se debe mostrar un mensaje al usuario que permita abandonar correctamente la aplicación.		

Tabla 3.7. RUC-07. Mensaje para abandonar la aplicación.

Identificador:	RUC-08	Necesidad:	Esencial
Título:	Personaje		
Descripción:	Se debe poder mover el personaje por la pantalla.		

Tabla 3.8. RUC-08. Personaje.

Identificador:	RUC-09	Necesidad:	Esencial
Título:	Pelotas		
Descripción:	Se deben poder mover las pelotas por la pantalla.		

Tabla 3.9. RUC-09. Pelotas.

Identificador:	RUC-10	Necesidad:	Esencial
Título:	Indicador de puntuación		
Descripción:	Se debe mostrar la puntuación acumulada durante la partida.		

Tabla 3.10. RUC-10. Indicador de puntuación.

Identificador:	RUC-11	Necesidad:	Esencial
Título:	Indicador de vidas.		
Descripción:	Se debe mostrar el número de vidas restante durante la partida.		

Tabla 3.11. RUC-11. Indicador de vidas.

Identificador:	RUC-12	Necesidad:	Esencial
Título:	Indicador de tiempo.		
Descripción:	Se debe mostrar el tiempo restante de la partida en el modo Tour.		

Tabla 3.12. RUC-12. Indicador de tiempo.

Identificador:	RUC-13	Necesidad:	Deseable
Título:	Indicador de pantalla		
Descripción:	Se debe poder mostrar el número de pantalla durante la partida en el modo Tour.		

Tabla 3.13. RUC-13. Indicador de pantalla.

Identificador:	RUC-14	Necesidad:	Deseable
Título:	Indicador de nivel.		
Descripción:	Se debe poder mostrar el numero de nivel durante la partida en el modo Panic		

Tabla 3.14. RUC-14. Indicador de nivel.

Identificador:	RUC-15	Necesidad:	Deseable
Título:	Sonido de explosión		
Descripción:	Se debe escuchar un sonido cuando se rompe una bola.		

Tabla 3.15. RUC-15. Sonido de explosión.

Identificador:	RUC-16	Necesidad:	Deseable
Título:	Sonido de bloque.		
Descripción:	Se debe escuchar un sonido cuando el jugador rompe un bloque.		

Tabla 3.16. RUC-16. Sonido de bloque.

Identificador:	RUC-17	Necesidad:	Deseable
Título:	Sonido de ítem.		
Descripción:	Se debe escuchar un sonido cuando el jugador obtiene un ítem.		

Tabla 3.17. RUC-17. Sonido de ítem.

Identificador:	RUC-18	Necesidad:	Deseable
Título:	Sonido fin de partida		
Descripción:	Se debe escuchar un sonido cuando el jugador pierde sus vidas y acaba la partida.		

Tabla 3.18. RUC-18. Sonido fin de partida.

Identificador:	RUC-19	Necesidad:	Deseable
Título:	Sonido victoria		
Descripción:	Se debe escuchar un sonido cuando el jugador gana la partida o concluya con éxito una pantalla.		

Tabla 3.19. RUC-19. Sonido victoria.

Identificador:	RUC-20	Necesidad:	Deseable
Titulo:	Sonido inicial		
Descripción:	Se debe escuchar un sonido al arrancar la aplicación.		

Tabla 3.20. RUC-20. Sonido inicial.

Identificador:	RUC-21	Necesidad:	Deseable
Título:	Icono en archivo ejecutable		
Descripción:	Se debe mostrar un icono en el ejecutable de la aplicación que la distinga del resto de aplicaciones.		

Tabla 3.21. RUC-21. Icono en archivo ejecutable.

Seguidamente se muestran los requisitos de usuario de restricción:

Identificador:	RUR-01	Necesidad:	Esencial
Título:	Plataforma Android		
Descripción:	El juego debe ser compatible con la plataforma Android		

Tabla 3.22. RUR-01. Plataforma Android.

Identificador:	RUR-02	Necesidad:	Deseable
Título:	Sonidos		
Descripción:	El jugador deberá poder subir/bajar el volumen del juego tocando los botones de volumen de su Smartphone.		

Tabla 3.23. RUR-02. Sonidos.

Identificador:	RUR-03	Necesidad:	Deseable
Título:	Formatos audio – imagen		
Descripción:	El formato para los ficheros de audio es .OGG y para las imágenes es .PNG		

Tabla 3.24. RUR-03. Formatos audio – imagen.

Identificador:	RUR-04	Necesidad:	Esencial
Título:	Acelerómetros		
Descripción:	El Smartphone debe ser compatible con el uso de acelerómetros.		

Tabla 3.25. RUR-04. Acelerómetros.

Identificador:	RUR-05	Necesidad:	Deseable
Título:	Vibración		
Descripción:	El Smartphone debe ser compatible con el uso de vibraciones.		

Tabla 3.26. RUR-05. Vibración.

Identificador:	RUR-06	Necesidad:	Esencial
Título:	Pantalla táctil		
Descripción:	El Smartphone debe ser compatible con el uso de la pantalla táctil.		

Tabla 3.27. RUR-06. Pantalla táctil.

3.1.3. Requisitos de software

Los requisitos de software describen el comportamiento del sistema desarrollado (requisitos funcionales) y el tipo de requisito (requisito de rendimiento, interfaz, operación, recursos, mantenimiento, calidad, seguridad, documentación, comprobación y aceptación de pruebas). Para esta aplicación no se han valorado todos los tipos de requisito.

Al igual que los anteriores requisitos de usuario, los requisitos de software vienen definidos por varios atributos:

- **Identificador**: Indica el tipo de requisito RS (Requisito software) añadiéndole una inicial que muestra si se trata de un requisito funcional (F), de rendimiento (R), de interfaz (I) o de recursos (Re) y un valor numérico. Este identificador debe ser un valor único.

- **Necesidad**: Muestra si la prioridad del requisito es Esencial o Deseable.

- **Título**: Frase breve que indica el nombre del requisito.

- **Fuente**: Muestra los requisitos de usuario en los que se basa este requisito software. Si no procede de ningún requisito de usuario y se incluye por consideración del analista, se identifica como ANA.

- **Descripción**: Breve comentario que describe el requisito.

A continuación se muestran los requisitos funcionales:

Identificador:	RSF-01	Necesidad:	Esencial
Título:	Lanzamiento del juego		
Fuente:	RUC- 01		
Descripción:	La aplicación deberá ser lanzada desde un fichero con extensión .apk generado al compilar e instalado previamente en el terminal.		

Tabla 3.28. RSF-01. Lanzamiento del juego.

Identificador:	RSF-02	Necesidad:	Esencial
Título:	Mostrar menú principal.		
Fuente:	RUC-01, RUC-04, RUC-05		
Descripción:	Al arrancar el juego, la aplicación deberá mostrar una pantalla inicial con las opciones Iniciar juego, Opciones e Instrucciones.		

Tabla 3.29. RSF-02. Mostrar menú principal.

Identificador:	RSF-03	Necesidad:	Esencial
Título:	Elección de modo de juego.		
Fuente:	RUC-01		
Descripción:	Cuando el usuario en el menú principal elija la opción Iniciar juego, se deberá mostrar una pantalla para elegir entre modo Panic y modo Tour.		

Tabla 3.30. RSF-03. Elección de modo de juego.

Identificador:	RSF-04	Necesidad:	Deseable
Título:	Mostrar diálogo pausa		
Fuente:	RUC-06		
Descripción:	Al pulsar el botón atrás del smartphone durante la partida, se deberá mostrar un dialogo, pausando el juego, permitiendo al usuario abandonar la partida.		

Tabla 3.31. RSF-04. Mostrar diálogo pausa.

Identificador:	RSF-05	Necesidad:	Esencial
Título:	Pantalla del juego		
Fuente:	RUC-08, RUC-09, RUC-10, RUC-11, RUC-12, RUC-13, RUC-14, RUR-04		
Descripción:	Al iniciar la partida, la pantalla principal deberá permitir el movimiento del personaje, el movimiento de las bolas de colores, mostrar la puntuación actual, el número de vidas restante, tiempo restante (solo en modo Tour) y la pantalla o el nivel actual según el modo de juego.		

Tabla 3.32. RSF-05. Pantalla del juego

Identificador:	RSF-06	Necesidad:	Esencial
Título:	Interacción con personaje		
Fuente:	RUR-06		
Descripción:	El smartphone debe ser compatible con el uso pantalla táctil para disparar las bolas de la pantalla.		

Tabla 3.33. RSF-06. Interacción con personaje.

Los requisitos funcionales de rendimiento están asociados al rendimiento del software en el smartphone, principalmente a la velocidad de procesado de nuestro juego.

Identificador:	RSR-01	Necesidad:	Deseable
Título:	Cargar juego		
Fuente:	ANA		
Descripción:	La aplicación debe tardar lo menos posible en ser cargada y mostrada al usuario.		

Tabla 3.34. RSR-01. Cargar juego.

Identificador:	RSR-02	Necesidad:	Deseable
Título:	Refrescar pantalla		
Fuente:	ANA		
Descripción:	La actualización de los datos de pantalla debe resultar imperceptible a ojos del usuario.		

Tabla 3.35. RSR-02. Refrescar pantalla

Identificador:	RSR-03	Necesidad:	Deseable
Título:	Movimientos en pantalla		
Fuente:	ANA		
Descripción:	Los movimientos de bolas y personaje deben ser suaves y no presentar movimientos bruscos.		

Tabla 3.36. RSR-03. Movimientos en pantalla

Identificador:	RSR-04	Necesidad:	Deseable
Título:	Interacción entre pantallas		
Fuente:	ANA		
Descripción:	La interacción entre las diferentes pantallas debe ser lo más rápida posible.		

Tabla 3.37. RSR-04. Interacción entre pantallas

Los requisitos de interfaz especifican como es la interfaz con la que el usuario deberá interactuar.

Identificador:	RSI-01	Necesidad:	Deseable
Título:	Imágenes del juego		
Fuente:	RUR-03		
Descripción:	Todas las imágenes del juego tienen formato .PNG		

Tabla 3.38. RSI-01. Imágenes del juego

Identificador:	RSI-02	Necesidad:	Deseable
Título:	Layouts		
Fuente:	ANA		
Descripción:	Todos los layouts de la aplicación están definidos en sus correspondientes archivos xml.		

Tabla 3.39. RSI-02. Layouts

Los requisitos de recursos indican de qué recursos físicos es necesario disponer para el desarrollo y ejecución de la aplicación.

Identificador:	RSRe-01	Necesidad:	Esencial
Título:	Entorno de desarrollo		
Fuente:	ANA		
Descripción:	El ordenador en el que se desarrollará el juego debe poseer el software necesario para el desarrollo de aplicaciones Android. No es obligatorio el uso de un IDE pero si muy recomendable.		

Tabla 3.40. RSRe-01. Entorno de desarrollo.

Identificador:	RSRe-02	Necesidad:	Esencial
Título:	Entorno de ejecución		
Fuente:	ANA		
Descripción:	El juego se debe ejecutar en un dispositivo que posea el sistema operativo Android.		

Tabla 3.41. RSRe-02. Entorno de ejecución.

3.1.4. Casos de uso

Los diferentes casos de uso sirven para identificar las relaciones existentes entre los actores (el jugador del juego en este caso) y el software, como muestra la figura 3.1.

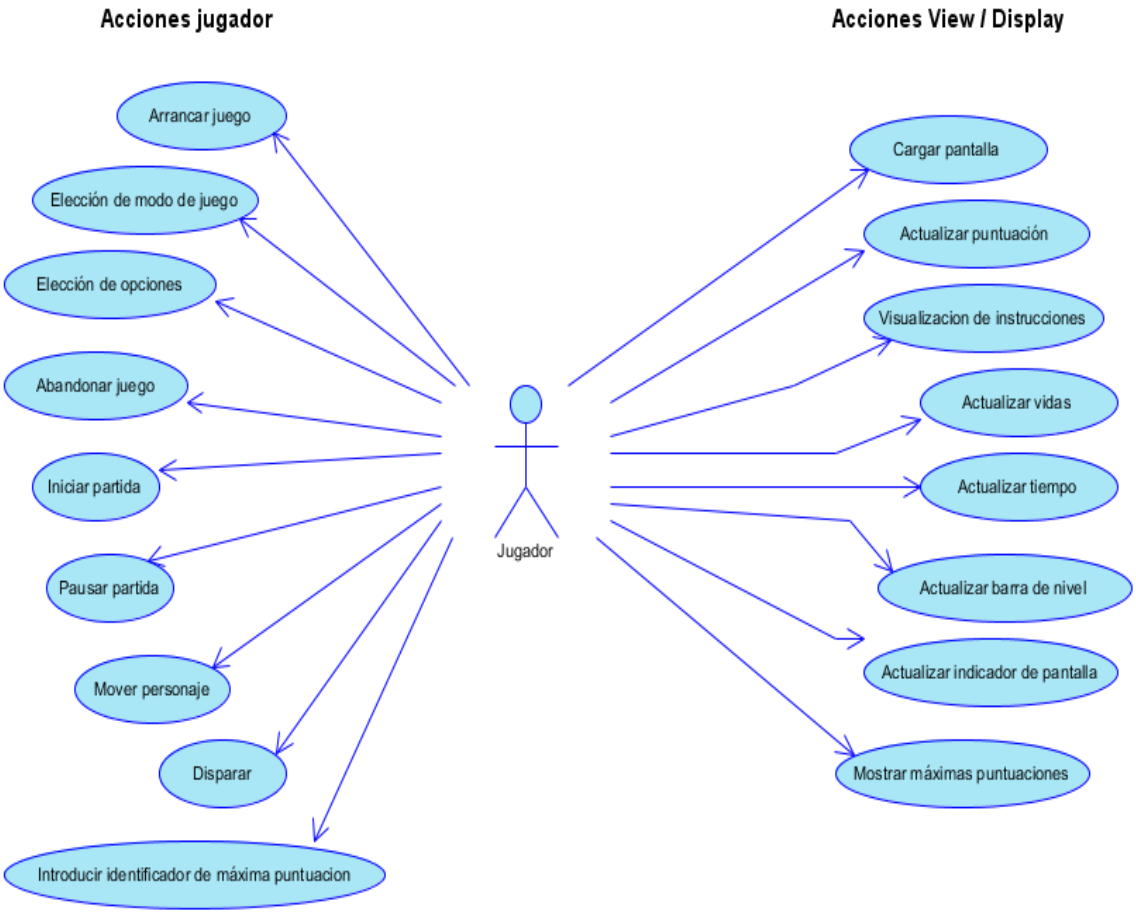


Figura 3.1. Diagrama de casos de uso.

Los casos de uso se representan mediante los siguientes atributos:

- **Identificador**: Muestra el caso de uso (CU) acompañado de un valor numérico. Este identificador debe ser un valor único.
- **Título**: Frase breve que identifica el nombre del caso de uso.
- **Actores**: Principales actores del caso de uso.
- **Objetivo**: Objetivo pretendido por el caso de uso.

- **Pre-condiciones**: Condiciones previas que deben cumplirse para que ocurra el caso de uso.

- **Post-condiciones**: Condiciones producidas a posteriori de que produzca el caso de uso.

A continuación se describen los distintos casos de uso:

Identificador:	CU-01
Título:	Arrancar juego
Actores:	Jugador
Objetivo:	Arrancar la aplicación.
Pre-condiciones:	Haber instalado el juego en el terminal.
Post-condiciones:	Se cargará el menú principal de la aplicación. En él, el usuario puede elegir si iniciar juego, elegir las opciones o leer las instrucciones.

Tabla 3.42. CU-01. Arrancar juego.

Identificador:	CU-02
Título:	Elección de modo de juego
Actores:	Jugador
Objetivo:	Elegir entre modo Tour y modo Panic.
Pre-condiciones:	Haber seleccionado iniciar juego en el menú principal.
Post-condiciones:	El juego cargará la modalidad elegida y se cargará la pantalla correspondiente esperando a que el jugador pulse la pantalla.

Tabla 3.43. CU-02. Elección de modo de juego.

Identificador:	CU-03
Título:	Elección de opciones de juego
Actores:	Jugador
Objetivo:	Elegir sonido activado/desactivado, vibración activada/desactivada y el número de vidas (1, 3 o 5).
Pre-condiciones:	Haber seleccionado opciones en el menú principal.
Post-condiciones:	La aplicación volverá al menú principal una vez que el jugador, tras elegir las opciones, pulse atrás en su terminal.

Tabla 3.44. CU-03. Elección de opciones de juego.

Identificador:	CU-04
Título:	Mostrar instrucciones
Actores:	Jugador
Objetivo:	Presentar al jugador las instrucciones para jugar al juego.
Pre-condiciones:	Haber seleccionado instrucciones en el menú principal.
Post-condiciones:	La aplicación volverá al menú principal una vez que el jugador pulse atrás en su terminal.

Tabla 3.45. CU-04. Mostrar de instrucciones.

Identificador:	CU-05
Título:	Abandonar juego.
Actores:	Jugador
Objetivo:	Mostrar al jugador un cuadro de diálogo permitiéndole abandonar el juego.
Pre-condiciones:	Haber pulsado atrás en el menú principal.
Post-condiciones:	Se cerrará la aplicación si el jugador elige Si y se volverá al menú principal si el jugador elige No.

Tabla 3.46. CU-05. Abandonar juego.

Identificador:	CU-06
Título:	Cargar pantalla
Actores:	Jugador
Objetivo:	Mostrar al jugador la pantalla inicial de la partida.
Pre-condiciones:	Elegir un modo de juego en la pantalla de selección de modo.
Post-condiciones:	El juego permanecerá parado hasta que el jugador toque la pantalla.

Tabla 3.47. CU-06. Cargar pantalla.

Identificador:	CU-07
Título:	Iniciar partida
Actores:	Jugador
Objetivo:	Comenzar el funcionamiento de la partida, permitiendo la interacción con el jugador.
Pre-condiciones:	Haber pulsado la pantalla una vez cargada la partida.
Post-condiciones:	Se iniciará la partida y el jugador podrá jugar una partida con normalidad.

Tabla 3.48. CU-07. Iniciar partida.

Identificador:	CU-08
Título:	Pausar partida
Actores:	Jugador
Objetivo:	Pausar el juego y mostrar un cuadro de diálogo.
Pre-condiciones:	Haber pulsado atrás en el terminal con la partida arrancada.
Post-condiciones:	El juego permanecerá parado hasta que el jugador seleccione en el cuadro de diálogo la opción No (volver al juego) o la opción Si (abandonar la partida y volver al menú principal).

Tabla 3.49. CU-08. Pausar partida.

Identificador:	CU-09
Título:	Mover personaje
Actores:	Jugador
Objetivo:	Mover el personaje por las zonas de la pantalla habilitadas (eje X)
Pre-condiciones:	Haber pulsado la pantalla después de mostrarla.
Post-condiciones:	El personaje se desplazará un valor constante en función del acelerómetro.

Tabla 3.50. CU-09. Mover personaje

Identificador:	CU-10
Título:	Disparar
Actores:	Jugador
Objetivo:	El personaje realizará un disparo.
Pre-condiciones:	Haber pulsado la pantalla táctil durante la partida
Post-condiciones:	Se lanza un disparo que puede provocar explosión de una bola o un bloque.

Tabla 3.51. CU-10. Disparar

Identificador:	CU-11
Título:	Actualizar puntuación
Actores:	Jugador
Objetivo:	Actualizar el valor de la puntuación de partida.
Pre-condiciones:	Haber iniciado partida y haber explotado una bola o bloque, o haber conseguido un ítem.
Post-condiciones:	Se actualiza la puntuación de la partida

Tabla 3.52. CU-11. Actualizar puntuación.

Identificador:	CU-12
Título:	Actualizar vidas
Actores:	Jugador
Objetivo:	Actualizar el número de vidas.
Pre-condiciones:	Haber iniciado partida y haber colisionado con una bola o haber conseguido el ítem de vida extra.
Post-condiciones:	Se actualiza el número de vidas de la partida.

Tabla 3.53. CU-12. Actualizar vidas.

Identificador:	CU-13
Título:	Actualizar tiempo de partida
Actores:	Jugador
Objetivo:	Actualizar el tiempo de la partida cada segundo.
Pre-condiciones:	Haber iniciado partida en modo Tour.
Post-condiciones:	Se actualiza el tiempo restante para la finalización de la partida.

Tabla 3.54. CU-13. Actualizar tiempo de partida.

Identificador:	CU-14
Título:	Actualizar barra de nivel
Actores:	Jugador
Objetivo:	Actualizar la barra de nivel.
Pre-condiciones:	Haber iniciado partida en modo Panic y haber explotado una bola.
Post-condiciones:	Se actualiza el nivel de la barra. Si la barra se llena, se aumenta el nivel de la partida y la barra vuelve a su estado inicial (vacía).

Tabla 3.55. CU-14. Actualizar barra de nivel.

Identificador:	CU-15
Título:	Actualizar indicador de pantalla
Actores:	Jugador
Objetivo:	Actualizar el número indicador de pantalla.
Pre-condiciones:	Haber iniciado partida en modo Tour y ganar en la pantalla actual
Post-condiciones:	Se añade 1 al valor actual de pantalla.

Tabla 3.56. CU-15. Actualizar indicador de pantalla.

Identificador:	CU-16
Título:	Mostrar máximas puntuaciones
Actores:	Jugador
Objetivo:	Mostrar pantalla con máximas puntuaciones.
Pre-condiciones:	Con la partida iniciada, haber perdido todas las vidas o haber llegado hasta el final del modo elegido.
Post-condiciones:	Se muestra una pantalla con las máximas puntuaciones.

Tabla 3.57. CU-16. Mostrar máximas puntuaciones.

Identificador:	CU-17
Título:	Introducir identificador de máxima puntuación
Actores:	Jugador
Objetivo:	Introducir 3 caracteres identificativos de la puntuación de la partida
Pre-condiciones:	Haber conseguido una de las cinco mejores puntuaciones finales.
Post-condiciones:	Se actualiza la pantalla de máximas puntuaciones. Al pulsar en la pantalla se vuelve al menú principal.

Tabla 3.58. CU-17. Introducir identificador de máxima puntuación

3.1.5. Diagrama de actividad del sistema

Los diagramas de actividad muestran la secuencia de actividades que sigue una aplicación mediante diagramas de flujo, desde el punto inicial hasta el punto final indicando muchas de las rutas de decisiones posibles durante la ejecución de la aplicación. Gracias a ellos se comprenden mejor las transiciones y los eventos del juego.

En la figura 3.2 se muestra el diagrama de actividades respectivo del juego.

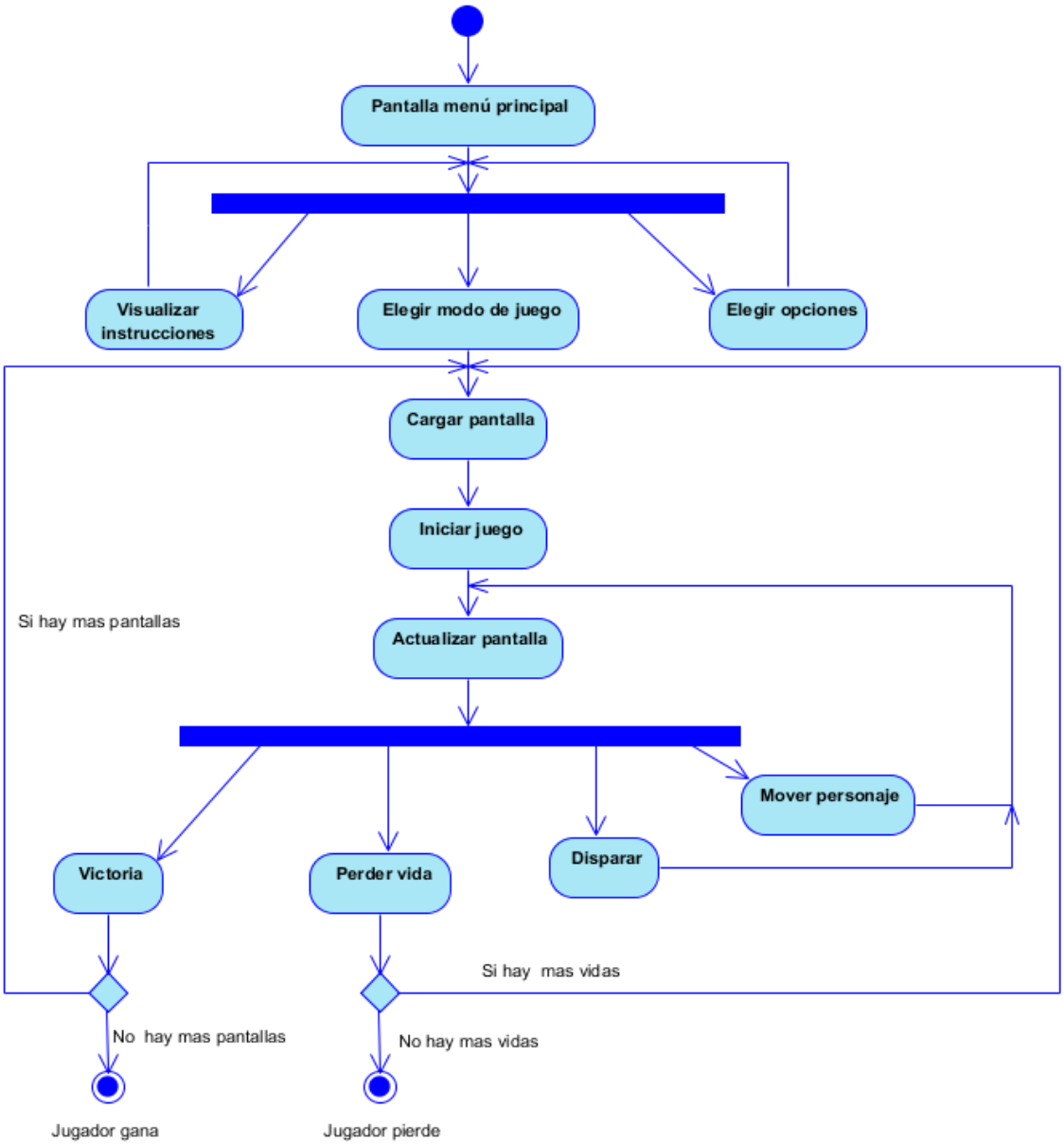


Figura 3.2. Diagrama de Actividades. Flujo del juego.

Al arrancar la aplicación se carga la pantalla del menú principal en la cual el jugador puede tomar tres decisiones: visualizar las instrucciones, ir a la pantalla de opciones o a la de selección de modo de juego. Las dos primeras regresan al menú principal tras cumplir su cometido. La tercera opción carga la partida una vez el jugador ha seleccionado el modo de juego deseado.

Con el juego cargado es necesario detectar el evento de pulsación de pantalla para iniciar el juego. En este punto se actualiza la pantalla constantemente a la vez que se pueden detectar diversos eventos. Dichos eventos pueden ser externos o internos a la aplicación.

Los eventos externos son provocados por el usuario como el movimiento del móvil detectado con el acelerómetro, el cual provoca el movimiento del personaje, y la pulsación de la pantalla, que provoca la realización de un disparo. En ambos casos se siguen actualizando los diferentes elementos de la pantalla.

Por otro lado, los eventos internos de la aplicación no son provocados por el usuario directamente sino que son consecuencia de la partida. Se trata de la detección de los sucesos que implican perder una vida y conseguir una victoria. En ambos casos existen dos posibilidades: que no existan más pantallas/vidas y se finalice la partida o que si existan más pantallas/vidas y se reinicie la partida actualizando los elementos necesarios.

3.1.6. Diagramas de secuencia

En los diagramas de secuencia se representa la interacción entre diferentes objetos de una aplicación a lo largo del tiempo para el cumplimiento de los casos de uso, mostrando los objetos que participan en la interacción y la secuencia de mensajes intercambiados entre sí.

La figura 3.3, representa el diagrama de secuencia general del juego.

La clase *SuperPang* se inicializa junto a la aplicación y muestra el menú principal. En él aparecen tres opciones: iniciar juego, opciones e instrucciones. Al pulsar sobre las dos últimas opciones se cargan las clases *Opciones* e *Instrucciones* respectivamente. La clase *Opciones* se encarga de mostrar al usuario varias opciones que puede activar, desactivar o modificar y la clase *Instrucciones* carga una ventana en la que se muestra al usuario las instrucciones para poder comprender el funcionamiento del juego.

Al pulsar sobre la opción iniciar juego, *SuperPang* llama a la clase *MenuPang* y el usuario puede elegir entre uno de los dos modos de juego. Si el usuario elige la opción situada a la izquierda, se llama a la clase *GamePanic* que se encarga de establecer como interfaz grafico una instancia de la clase *GameViewPanic*. Si por el contrario el usuario elige la opción derecha del menú, se llama a la clase *GameTour*. Esta clase es la responsable cargar una instancia de la clase *GameViewTour* como interfaz gráfico.

Sea cual sea la clase cargada, el procedimiento es el mismo. La partida inicialmente se mantiene en pausa hasta que el usuario decida iniciarla pulsando la pantalla. La clase cargada como interfaz gráfico es la encargada de pintar la pantalla, actualizar la posición del personaje y las pelotas y guardar toda la información referida a puntuación, numero de vidas, etc. También se encarga de detectar las pulsaciones que el usuario realiza en la pantalla y determinar la acción a realizar así como de comunicarse con *GameTour* o *GamePanic* cuando detectan un suceso, bien perder una vida o bien salir victorioso de la pantalla actual.

En este último paso, dichas clases se comunican con la instancia de la clase establecida como interfaz a través de mensajes mediante un objeto Handler que debe gestionar la siguiente pantalla a mostrar. Cuando el suceso perder vida significa el fin de la partida, este objeto se encarga de llamar a las clases *RankingTour* o *RankingPanic* respectivamente, las cuales nos muestran el ranking de puntuaciones del modo de juego.

Estas dos clases permanecen visible hasta que el usuario decida tocar la pantalla, momento en el cual se destruyen y el control vuelve al menú principal.

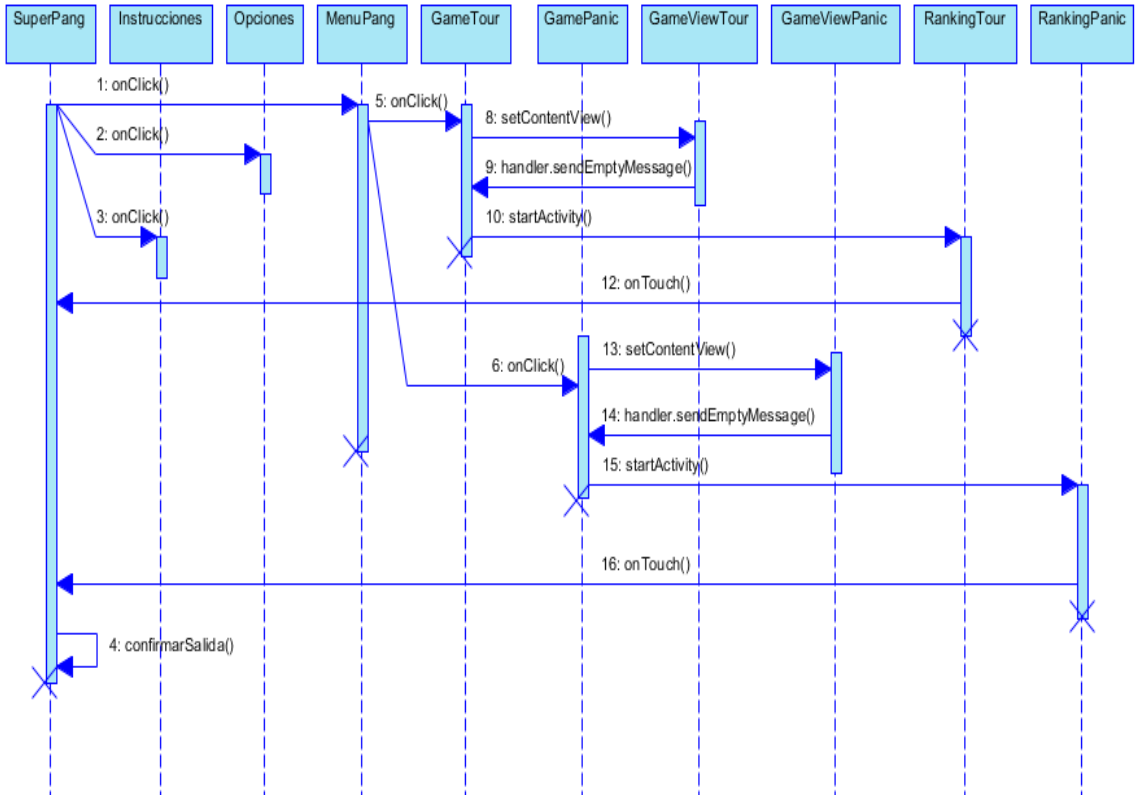


Figura 3.3. Diagrama de secuencia. General.

La figura 3.4 representa el diagrama de secuencia del juego para poder mostrarlo con más detalle que en el diagrama anterior. Se muestra el flujo de trabajo del modo Tour aunque el procedimiento es aplicable al modo Panic.

La clase *GameTour* se encarga de instanciar la clase *GameViewTour*, que hereda de *SurfaceView*, y la establece como interfaz gráfico con la llamada a setContentView(). Además inicializa su estado a Ready, a la espera de interacción con el jugador, con el método setState(Ready). También es responsable de instanciar la clase *MusicaDeFondo*, que hereda de la clase Service, con startService(). Esta clase se encarga de reproducir la música de fondo sin interacción con el jugador. Inicialmente se encuentra creado el servicio pero no reproduce música hasta que el usuario no comience el juego.

En el constructor de *GameViewTour* se inicializan las variables de la clase *Efectos* con initEfectos() para posteriormente cargar los efectos de sonido con la llamada addEfecto(), y poder reproducirlos durante la partida. También se instancia la clase *MiThreadTour* que será quien maneje el interfaz gráfico de la aplicación. Por herencia se llama automáticamente a los métodos surfaceCreated(), que se encarga de iniciar el hilo principal, y surfaceChanged(), encargado de obtener las dimensiones de la pantalla y llamar al método inicializarVariables() que sirve para inicializar todas las variables necesarias de la partida.

Para que el juego se arranque y comience la partida, el usuario debe tocar la pantalla, cuya pulsación es detectada en el método onTouchEvent() de la clase *GameViewTour*. Este método es el encargado de dos funciones disyuntivas: si el juego se encuentra en estado Ready se cambia a estado Running, comenzando la partida y la música de fondo o si el juego ya se encontraba en estado Running, se realiza la acción de disparar llamando al método activarDisparo() de la clase *Disparo*.

El hilo principal se encarga de pintar todos los objetos de pantalla con la llamada a onDraw(Canvas). En dicha llamada se pintan todos los elementos del juego incluyendo personaje, bolas y bloques necesarios. El hilo principal también se encarga de actualizar la posición del personaje en función del acelerómetro llamando a update(int) de la clase *Personaje* y las posiciones de todas las bolas del juego con el método actualizarPosiciones().

El hilo principal depende del estado en que se encuentre la partida. Si el estado es Ready o Pause, simplemente se encarga de pintar la pantalla mientras que si el estado es Running permite la interacción del usuario con la aplicación.

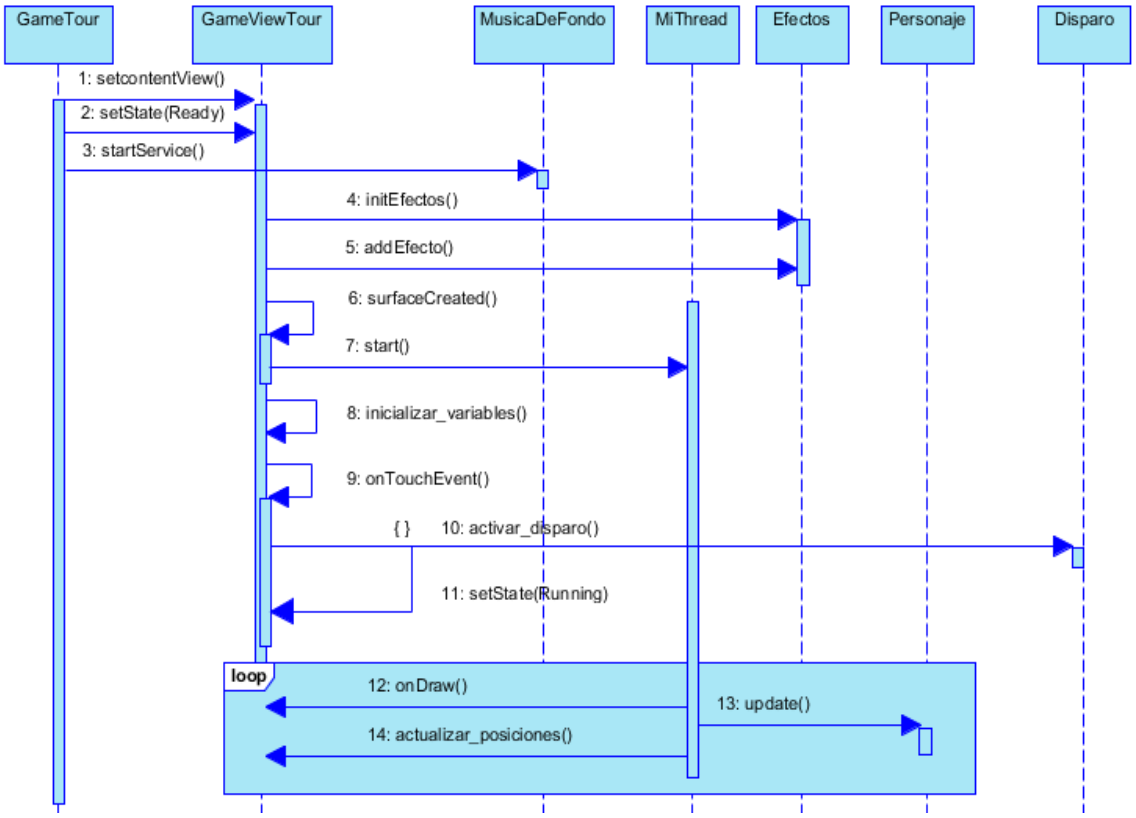


Figura 3.4. Diagrama de secuencia. Iniciar partida.

3.2. Diseño

Tras realizar el análisis y evaluar qué se quiere hacer, se pasa a la fase de diseño en la que se evalúa cómo se quiere hacer. El diseño debe realizarse sobre todos los componentes de la aplicación y se debe conseguir que todos los aspectos queden completamente definidos de cara a la siguiente fase del proyecto.

El objetivo es realizar un diseño correcto para evitar rectificaciones en la fase de implementación y por lo tanto cumplir los plazos iniciales de entrega del proyecto.

Este apartado se ha dividido en cuatro sub-secciones:

- **Arquitectura**: Definición a alto nivel de la estructura del juego.
- **Interfaces**: Muestra los diseños iniciales realizados para cada pantalla del juego.
- **Carpetas del proyecto**: Muestra el árbol de carpetas de la aplicación.
- **Diagrama de clases**: Muestra las clases y las relaciones entre ellas.

3.2.1. Arquitectura

La arquitectura [17] muestra un diseño a alto nivel de la estructura del juego, en el que se debe definir los módulos principales de la aplicación, las responsabilidades de cada uno, la interacción entre ellos y el control y flujo de datos.

La arquitectura del juego (figura 3.5) está compuesta por dos módulos principales, el Core de la aplicación y el módulo de Entrada/Salida. El primero debe encargarse de la creación de todos los componentes y sus ciclos de vida y el segundo se encarga de permitir al usuario interactuar con dichos componentes y mostrar las diferentes vistas de la aplicación.

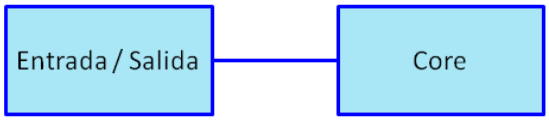


Figura 3.5. Arquitectura del juego. Diagrama de módulos.

3.2.2. Interfaces

En esta sección se muestran los diferentes diseños iniciales que fueron realizados para cada pantalla de la aplicación.

Todas las pantallas de la aplicación se han forzado a mostrarse en posición horizontal por dos motivos. El primero se debe al tamaño limitado de las pantallas. La mayoría de los smartphones poseen una pantalla alargada y eso provoca que este juego se pueda aprovechar mejor horizontalmente. El segundo motivo está relacionado con el primero, ya que nuestro personaje se va a desplazar a lo largo del eje horizontal de la pantalla y para ello es mejor colocar el móvil en posición horizontal para disponer de más recorrido de movimiento.

La figura 3.6 muestra el diseño inicial de la pantalla del menú principal, mostrado nada más arrancar la aplicación en el terminal. En dicha pantalla aparece la imagen del juego en la zona superior y debajo las diferentes opciones en forma de botón.

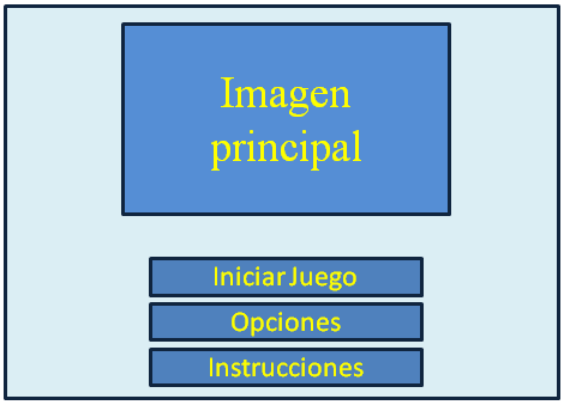


Figura 3.6. Interfaces. Menú principal.

Si el usuario elige iniciar juego, se muestra el diseño de la pantalla de elección de modo de juego en la que el usuario podrá elegir entre dos opciones de juego representadas por dos imágenes y una breve descripción (figura 3.7). En pantalla también se muestra un indicador de tiempo, que representa el tiempo restante para pasar a la siguiente pantalla si el usuario no elige.

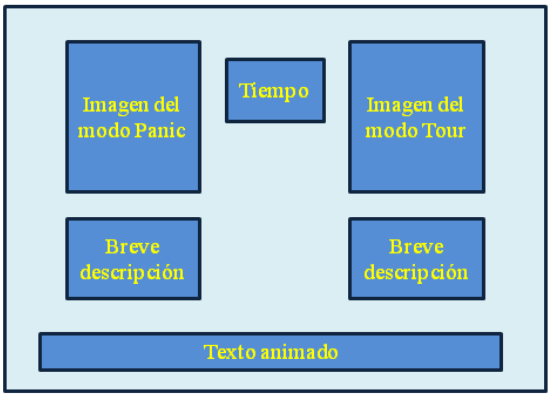


Figura 3.7. Interfaces. Elección de modo

Si la opción elegida es opciones, se muestra una pantalla en la que el usuario puede activar/desactivar el sonido y la vibración del juego así como elegir el número de vidas iniciales de la partida (figura 3.8). Ambas opciones tienen debajo del título una breve descripción.

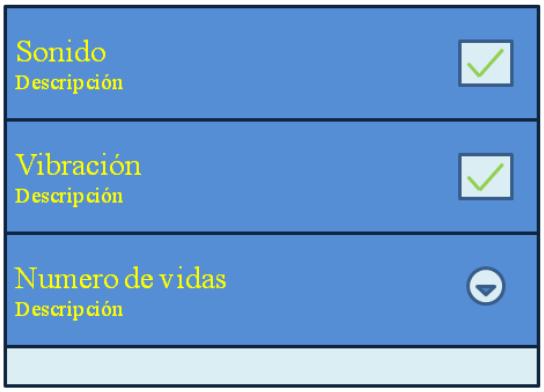


Figura 3.8. Interfaces. Opciones

La última de las opciones del menú principal, instrucciones, muestra un scroll con las instrucciones generales y de cada modo de juego (figura 3.9) de manera que cualquier usuario pueda empezar a jugar y entender el funcionamiento leyéndolas.

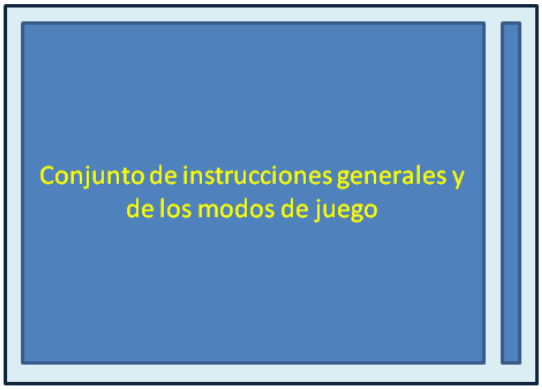


Figura 3.9. Interfaces. Instrucciones.

Para el diseño de la partida se ha dividido la pantalla en dos secciones. En la parte superior ocurrirá toda la acción del juego y en la zona inferior se encuentran todas las puntuaciones e indicadores del juego. El jugador podrá utilizar toda la pantalla para interactuar con el personaje. La figura 3.10 muestra el diseño del modo Tour y la figura 3.11 muestra el diseño del modo Panic.

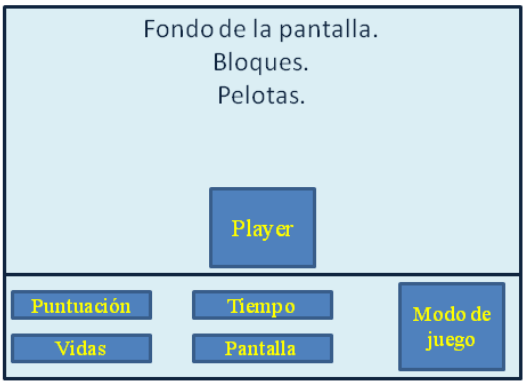


Figura 3.10. Interfaces. Modo Tour

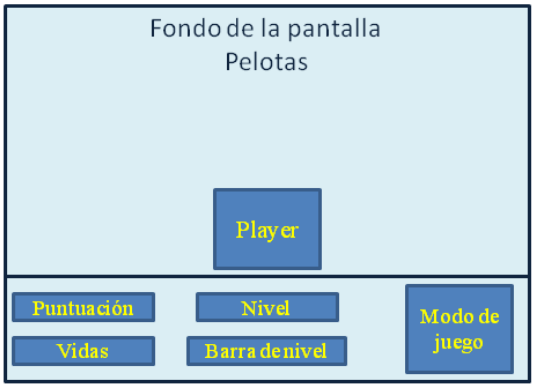


Figura 3.11. Interfaces. Modo Panic.

Otro diseño realizado fue la advertencia al usuario antes de abandonar el juego (figura 3.12). Muestra un cuadro de diálogo preguntando si quiere abandonar el juego y el usuario solo debe pulsar sobre los botones Sí o No según sea su decisión. Esta vista se ha utilizado también en el menú principal a la hora de preguntar al usuario antes de abandonar la partida.

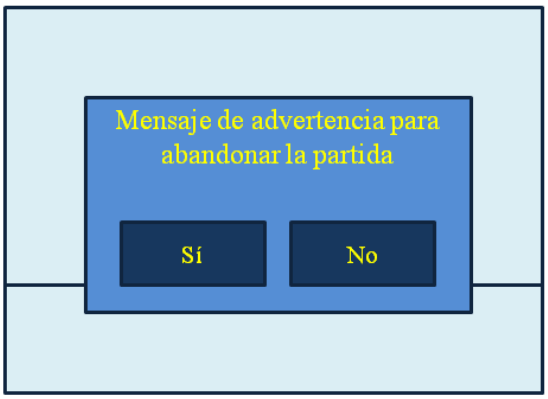


Figura 3.12. Interfaces. Abandonar partida.

La última de las pantallas que se muestra al usuario durante la partida es la pantalla de puntuaciones (figura 3.13), en la cual aparecen representados los cinco primeros puestos con sus respectivas puntuaciones, su identificador de puntuación y la pantalla o nivel alcanzado en la partida.

Figura 3.13. Interfaces. Máximas puntuaciones.

3.2.3. Carpetas del proyecto

Cuando se crea un proyecto Android en Eclipse se genera automáticamente una estructura en carpetas [18] que sirve para poder generar la aplicación a posteriori.

La organización de las carpetas de la aplicación se muestra en la figura 3.14.

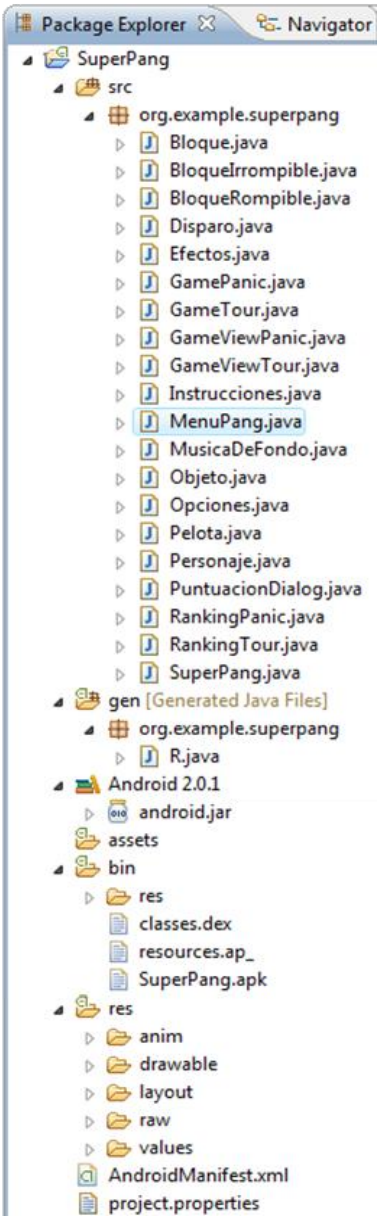


Figura 3.14. Carpetas del proyecto. Estructura

La carpeta **/src** incluye todo el código fuente creado para la aplicación.

La carpeta **/gen** contiene una serie de elementos de código que son generados automáticamente cada vez que se compila el proyecto. En el presente proyecto tan solo se ha generado el fichero R.java, el cual contiene una serie de constantes con los identificadores de los recursos almacenador en la carpeta **/res** y para acceder a ellos a través de este identificador.

La carpeta **Android 2.0.1** contiene todas las librerías requeridas para la aplicación. La numeración 2.0.1 corresponde a la versión SDK necesaria.

La carpeta **/assets**, no usada en este juego, se utiliza para el almacenamiento de recursos externos con formato .raw como ficheros de configuración, de datos, etc. Se accede a ellos a través de su ruta como cualquier otro recurso del sistema.

La carpeta **/bin** es el directorio de salida donde se puede encontrar el archivo instalable con formato .apk y otros recursos compilados.

La carpeta **/res** se descompone a su vez en un conjunto de subcarpetas que sirven para separar los diferentes recursos externos de la aplicación. En **res/anim/** se encuentran los archivos xml que sirven para dar animación a las vistas. La subcarpeta **res/drawable/** contiene todas las imágenes de la aplicación. En la ruta **res/layout/** se encuentran los archivos xml que definen los diferentes interfaces gráficos de la aplicación. En **res/raw/** se encuentran todos los ficheros de audio y efectos de sonido utilizados en el juego. Por último, **res/values/** es la carpeta que contiene otros archivos xml, donde se declaran strings, colors o arrays, que pueden ser referenciados en el código de la aplicación.

Fuera de estas carpetas se encuentra el fichero **AndroidManifest.xml** que describe la naturaleza de la aplicación y sus componentes. En él se deben incluir también todos los permisos que requieren la aplicación o el nivel API requerido, entre otros datos. También aparece el fichero **project.properties** que muestra las propiedades del proyecto.

3.2.4. Diagrama de clases

Los diagramas de clases son un tipo de diagramas que muestran las clases de un sistema, sus atributos y las relaciones entre ellas.

Cada una de las clases que contienen interfaz gráfica y permiten interacción con el usuario extiende de la clase Activity. Estas clases deben declararse en el fichero AndroidManifest.xml. Una Activity es un componente de aplicación que proporciona una interfaz con la cual el usuario puede interactuar y hacer algo. Una aplicación puede tener definidas muchas actividades pero el usuario solo puede interactuar con una a la vez. También deben declararse en el fichero xml las clases que extiendan de Service, ContentProvider o BroadcastReceiver, aunque en el presente proyecto no se han utilizado las dos últimas.

Además de las actividades se han definido otras clases auxiliares. Algunas de ellas extienden de SurfaceView y son utilizadas como interfaz gráfico mientras que otras no contienen interfaz pero son necesarias para el funcionamiento correcto de la aplicación.

Las figuras 3.15 y 3.16 muestran todas las clases utilizadas en este proyecto. Además de los métodos señalados, se han implementado métodos gets y sets para acceder a los atributos privados de la clase.

Por motivos de espacio se ha decidido dividir el diagrama de clases en dos figuras. La línea dorada relaciona las clases GameTour y GameViewTour, la línea de color rosa relaciona GamePanic y GameViewPanic, la línea roja une GameViewTour y MusicaDeFondo y la línea verde une GameViewPanic con MusicaDeFondo.

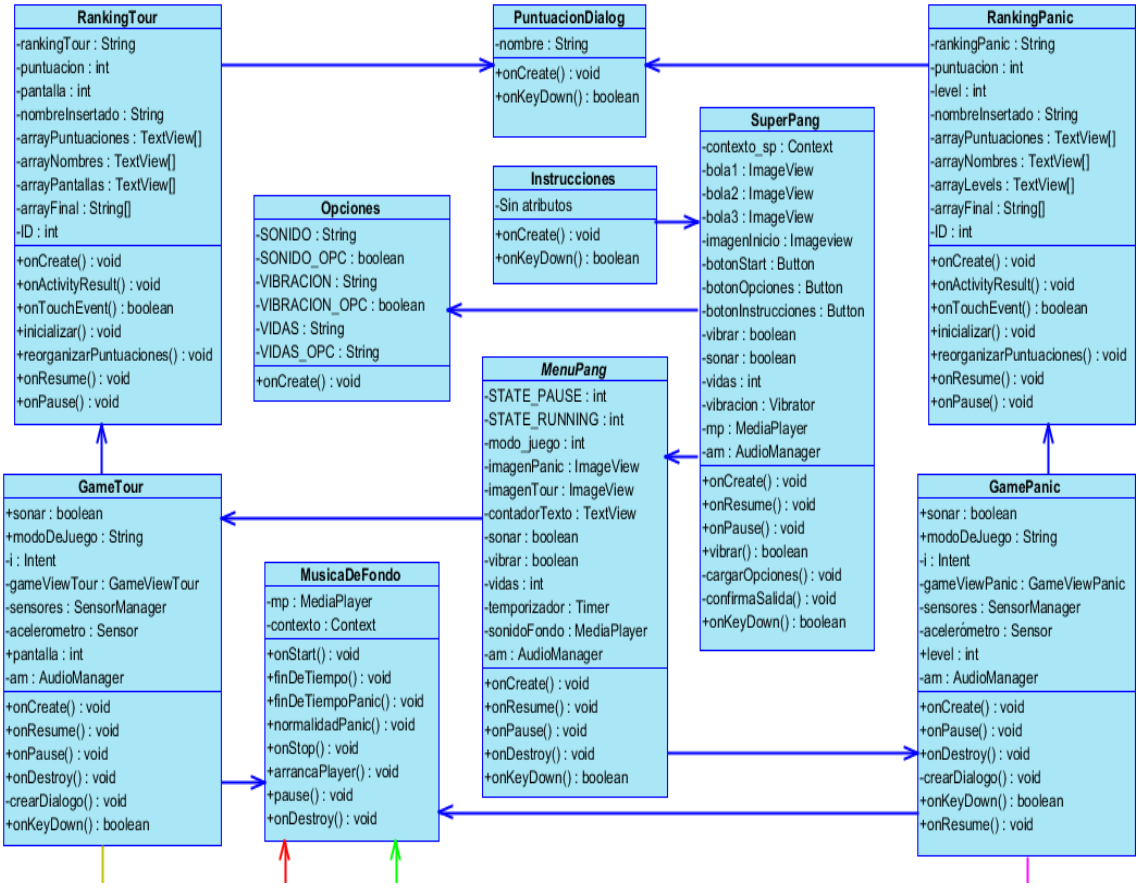


Figura 3.15. Diagrama de Clases. Parte I.

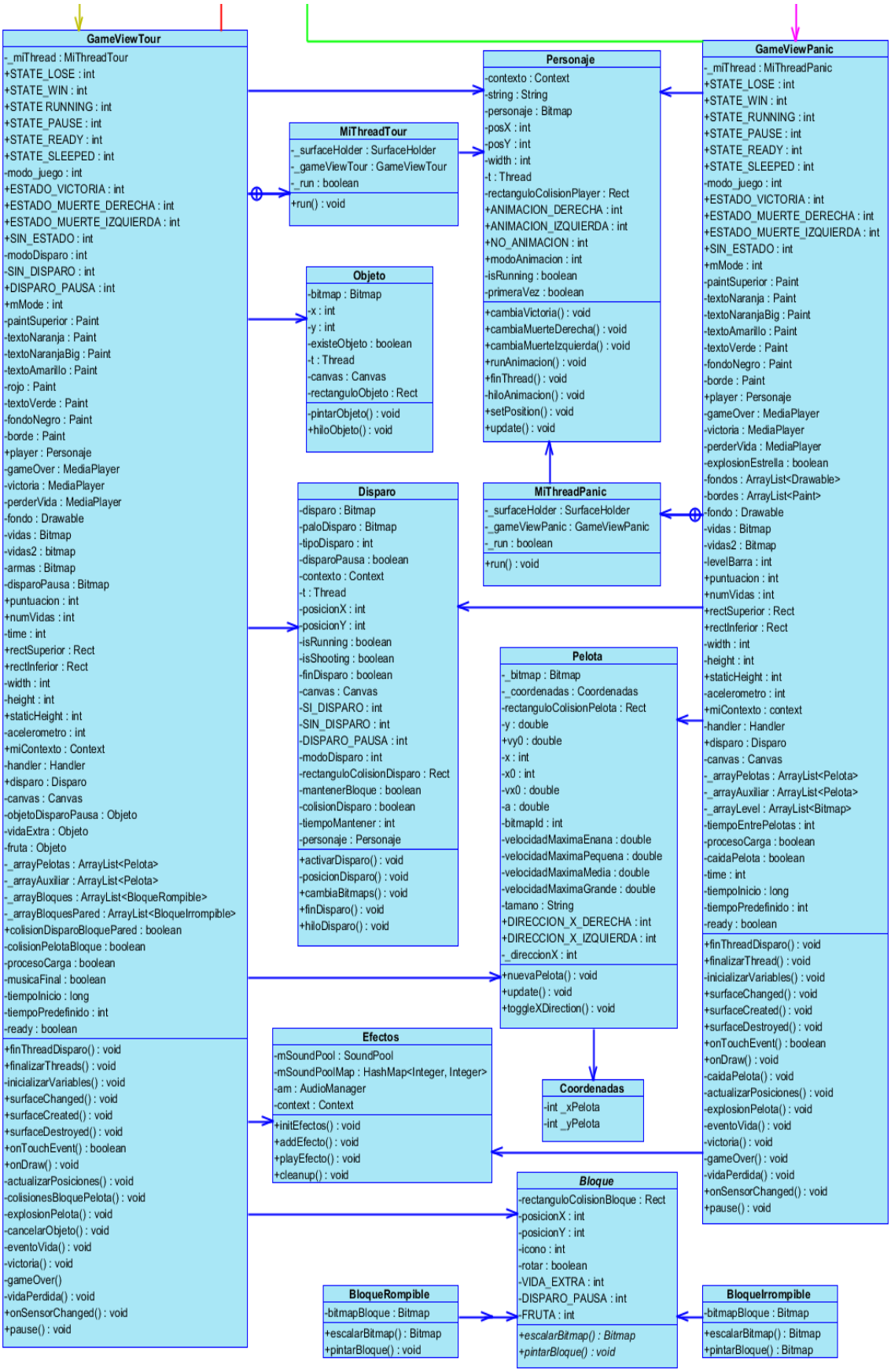


Figura 3.16. Diagrama de Clases. Parte II.

A continuación se detallan las diferentes clases de la aplicación.

La clase **SuperPang** (tabla 3.59) es la clase inicial de nuestra aplicación y muestra el menú principal con las tres opciones posibles: iniciar juego, opciones e instrucciones. Entre sus miembros se encuentran el Context de la aplicación, variables Button, ImageView e ImageButton que representan las diferentes opciones a elegir, las variables que almacenan el valor de las opciones del juego, un MediaPlayer que reproduce un sonido inicial y un AudioManager para manejar el volumen de la aplicación. El método onKeyDown() detecta el movimiento entre las opciones, la pulsación de las tecla de volumen y de la tecla back, mostrando en este caso un cuadro de dialogo para que el usuario confirme su deseo de abandonar la aplicación, en cuyo caso se accede al método confirmaSalida(). Inicialmente, las opciones del juego son cargadas en cargarOpciones() y se actualizan cada vez que la aplicación llama al método onResume(). En el método onCreate() se implementan los escuchadores para cada uno de los botones del menú principal de manera que al pulsar sobre ellos se carga la actividad correspondiente pasándole además las opciones del juego.

Clase:	SuperPang	Herencia:	Activity
Miembros:	Context contexto_sp ImageView bola1, bola2, bola3, imagenInicio Button botonStart, botonOpciones, botonInstrucciones boolean vibrar, sonar int vidas Vibrator vibracion MediaPlayer mp AudioManager am		
Métodos:	void onCreate(Bundle savedInstanceState) static boolean vibrar(boolean permiso, int tipo) void cargarOpciones() void confirmaSalida() void onResume() boolean onKeyDown(int keyCode, KeyEvent event)		

Tabla 3.59. Clase SuperPang. Miembros y métodos.

La clase **Opciones** (tabla 3.60) representa una actividad que muestra tres opciones modificables por el usuario: sonido, vibración y número de vidas (figura 3.8). Tiene como miembros las variables para almacenar dichas preferencias. En el método onCreate() se carga el archivo xml con las opciones, a las cuales se accede a través de métodos gets.

Clase:	Opciones	Herencia:	PreferenceActivity
Miembros:	String SONIDO, VIBRACION boolean SONIDO_OPC, VIBRACION_OPC String VIDAS, VIDAS_OPC		
Métodos:	void onCreate(Bundle savedInstanceState) boolean getSonido(Context context) boolean getVibracion(Context context) int getVidas(Context context)		

Tabla 3.60. Clase Opciones. Miembros y métodos.

La clase **Instrucciones** (tabla 3.61) no tiene ningún miembro. El método onCreate() carga el xml correspondiente que muestra un ScrollView con el texto de las instrucciones (figura 3.9) y en el método onKeyDown() se detecta la pulsación del botón back, se cierra la actividad y se vuelve al menú principal.

Clase:	Instrucciones	Herencia:	Activity
Miembros:			
Métodos:	onCreate(Bundle savedInstanceState) onKeyDown(int keyCode, KeyEvent event)		

Tabla 3.61. Clase Instrucciones. Miembros y métodos.

La clase **MenuPang** (tabla 3.62) muestra una actividad con dos imágenes y un contador. Cuando el usuario elija una de las opciones se cargara la modalidad de juego correspondiente. Tiene como miembros dos ImageView que representan las imágenes de las modalidades de juego (figura 3.17), un TextView para actualizar el contador de tiempo junto a un Timer que lo controla, tres variables int para manejar el estado de la actividad, dos boolean y otro int para tener constancia de las vidas iniciales elegidas por el usuario, un objeto MediaPlayer para reproducir la música de esta actividad y un objeto AudioManager para el control del volumen.

a) Modo Panic b) Modo Tour

Figura 3.17. MenuPang. Imágenes.

En el método onCreate() donde se carga la vista, se implementan los escuchadores de las imágenes y se crea el Timer que actualiza el TextView cada segundo. Cuando el

usuario seleccione una imagen y pulse sobre ella se cargará la actividad correspondiente en el método onKeyDown(). Si el usuario elige la opción derecha entra en modo Tour y se carga la clase GameTour, y si elige la opción izquierda entra en modo Panic y se carga la clase GamePanic. Con los métodos getState() y setState(int state) se controla el estado de la aplicación de manera que si el sistema llama a onPause() u onResume() la aplicación pueda pausar o recuperar tanto la música como el contador. El método onDestroy(), llamado al destruir la actividad, sirve para parar completamente la música.

Clase:	MenuPang	Herencia:	Activity
Miembros:	int STATE_PAUSE, STATE_RUNNING int modo_juego ImageView imagenPanic, imagenTour TextView contadorTexto boolean sonar, vibrar int vidas Timer temporizador MediaPlayer sonidoFondo AudioManager am		
Métodos:	void onCreate(Bundle savedInstanceState) void setState(int state); int getState() void onResume() void onPause() void onDestroy() boolean onKeyDown(int keyCode, KeyEvent event)		

Tabla 3.62. Clase MenuPang. Miembros y métodos.

La clase **GameTour** (tabla 3.63) añade un objeto GameViewTour a la actividad, que se encarga de pintar y manejar la partida. Además controla el avance o reinicio de pantallas de la partida mediante un objeto Handler. Tiene como miembros un boolean para manejar la preferencia de sonido, un Intent para lanzar el Service que modela la música de la partida, el citado objeto GameViewTour, un objeto SensorManager y otro Sensor para registrar el acelerómetro a usar en la partida. También tiene un int que representa el número de pantalla del juego, un objeto AudioManager que controla el volumen durante el juego, y un String usado en MusicaDeFondo para diferenciar los diferentes modos de juego.

En cuanto a métodos, en onCreate() se crea el objeto Handler para intercambiar mensajes con GameViewTour. Además se crea, si no está creado, el fichero de puntuaciones máximas, se instancia la clase GameViewTour y registra el sensor acelerómetro en dicha instancia, que será el responsable de manejar el sensor. También se crea el servicio para la música de fondo. En onKeyDown() se detecta la pulsación de la tecla back y llama al método crearDialogo() en el cual el usuario podrá abandonar la partida (figu-

ra 3.12). El método onPause(), pausa la música y muestra el diálogo anterior, mientras que onDestroy() para completamente el servicio, finaliza los hilos activos de la aplicación y vuelve al menú principal.

En el objeto Handler se determina cuándo hay que reiniciar la pantalla y cuando avanzar a la siguiente, evaluando si hay que cargar una pantalla nueva (actualizando datos de GameViewTour) o si se ha finalizado la partida y se debe cargar la pantalla de máximas puntuaciones, en cuyo caso se carga la clase RankingPanic.

Clase:	GameTour	Herencia:	Activity
Miembros:	boolean sonar String modoDeJuego Intent i GameViewTour gameViewTour SensorManager sensores Sensor acelerometro int pantalla AudioManager am		
Métodos:	void onCreate(Bundle savedInstanceState) void onResume() void onPause() void onDestroy() void crearDialogo() boolean onKeyDown(int keyCode, KeyEvent event)		

Tabla 3.63. Clase GameTour. Miembros y métodos.

La clase **GameViewTour** (tabla 3.64) hereda de SurfaceView e implementa el interfaz SurfaceHolder.Callback, el cual permite manejar los cambios que suceden en el SurfaceView. Esta clase es la encargada de controlar la partida del modo Tour y solo cede el control a GameTour cuando se pierde una vida o se consigue una victoria. Como miembros cuenta con el hilo principal de la partida (objeto de la clase MiThreadTour), el tipo de disparo y el bitmap del jugador cuando gana o pierde, variables relacionadas con la apariencia de la pantalla y textos, variables boolean para detectar colisiones, variables Objeto que se pueden crear cuando se rompa un bloque y representan los ítems que el jugador puede coger, varios ArrayList para almacenar los bloques y las pelotas de la pantalla, un objeto Canvas sobre el que se pintará todo el juego, un objeto Disparo para representar los disparos y variables que controlan el estado del juego.

Inicialmente la partida se encuentra en estado Ready, y cuando el usuario pulsa la pantalla por primera vez se pasa a modo Running. En el primer caso solo se pinta la pantalla a la espera de comenzar la partida mientras que en el segundo, la partida comienza, actualizándose la posición del personaje y las pelotas de la pantalla e iniciando

la música de la partida. Si el usuario pulsa la tecla back se entra en modo Pause, mostrando el diálogo de la clase GameTour (figura 3.12), mientras que si decide salir al escritorio o se recibe una llamada, deja de mostrarse la aplicación, y se pasa al estado Sleeped, en el cual, la aplicación esta pausada pero activa.

En el constructor se inicializan las variables que no van a cambiar durante toda la partida como son el hilo principal que maneja el canvas, los efectos de sonido, el objeto Handler, los sonidos de victoria o vida perdida y los bitmaps con los que el personaje no interactúa.

Al iniciarse la clase, como hereda de SurfaceView, tras la llamada al constructor, el sistema llama automáticamente a surfaceCreated() y surfaceChanged(). En cuanto al funcionamiento del SurfaceView, solo existe mientras este visible, y cuando deja de estarlo se destruye, llamando a surfaceDestroyed(). Por este motivo se han decidido sobrescribir los tres métodos, de manera que en ellos se manejen los estados de la aplicación, sin afectar a los hilos del programa.

El método surfaceCreated(), es el encargado de inicializar el hilo principal (o si ya existía, entrar en modo pause) y surfaceChanged(), toma las medidas de la pantalla y llama a inicializar_variables() si la partida aun no ha comenzado. Este método, llamado cada vez que se inicia una pantalla, se encarga de inicializar todas las variables necesarias para la interacción durante la partida como son los rectángulos que dividen la vista en dos, los Paint que definen los textos, los arrays para el almacenamiento de objetos Bloque y Pelota, el objeto Player que representa el personaje, el fondo y borde elegido en función del número de pantalla y la configuración de bloques y pelotas, también en función del número de pantalla. Por último surfaceDestroyed() solamente establece el estado de la aplicación como Sleeped para su posterior recuperación.

El hilo principal se mantiene activo durante la duración de la actividad GameTour, y a través de los métodos getState() y setState(int state) se controla el ciclo de vida del SurfaceView de manera que solo durante el estado Running el usuario puede jugar. Durante el estado Pause, se muestra un cuadro de dialogo que permite al usuario abandonar la partida, en estado Ready se encuentra a la espera de que el usuario decida iniciar partida y en estado Sleeped la actividad no se encuentra visible pero permanece activa.

Las pulsaciones de pantalla son detectadas en onTouchEvent(), quien decidirá si se debe arrancar la partida (cambiando el estado del juego a Running) o realizar un disparo. El método onDraw() se encarga de pintar por pantalla el fondo, los textos, el personaje, las pelotas y los bloques. Además durante los procesos de carga de pantalla, en este método, solo muestra una pantalla en negro y cuando ocurre un evento de victoria o vida perdida se encarga de llamar a los métodos victoria() o eventoVida().

Con el método actualizarPosiciones() se actualizan las posiciones de todas las pelotas y bloques y se detectan las colisiones entre los diferentes objetos de la partida. En

algunos casos, dichas colisiones son tratadas dentro de éste método y en otros como las colisiones pelota-bloque y pelota-disparo se recurre los métodos colisionPelotaBloque() y explosionPelota(). El primero hace rebotar a la pelota y el segundo rompe la pelota y crea dos pelotas nuevas de un tamaño inferior aumentando a su vez la puntuación de la partida.

El movimiento del personaje, manejado por el acelerómetro se gestiona en el método onSensorChanged(), que según la orientación del móvil asigna un valor constante al desplazamiento del personaje y controla el modo de animación del mismo.

Cuando se detecte que no hay más pelotas en pantalla, se llama al método victoria(), emitiendo un sonido característico para tal situación, mostrando un texto en pantalla indicando que se ha ganado la pantalla y mediante el objeto Handler se comunica con la clase GameTour quien decidirá si avanzar a la siguiente pantalla o mostrar una pantalla de máximas puntuaciones.

Si por el contrario lo que se detecta es una colisión entre el personaje y una de las pelotas, se resta una vida, se decide si se reinicia la partida (llamando a vidaPerdida()) o si se finaliza (llamando a gameOver()), se emite un sonido para la situación, y se lo comunica a GameTour.

Tanto en victoria() como en perderVida() y gameOver() se detecta si hay algún ítem activo (instancia de la clase Objeto) y procede a parar su hilo y eliminar el objeto para que no influya en la próxima pantalla.

Por último, esta clase implementa dos métodos, finThreadDisparo() y finalizarThreads(), que sirven para acabar con los hilos activos de la aplicación. El primero de los métodos solamente afecta al objeto Disparo, ya que corre en su propio hilo, y es el único que se destruye en cada partida. Este método se llama cada vez que se pierde una vida o se supera una pantalla. El segundo método, solamente se llama cuando el usuario abandona la partida voluntariamente y se encarga de finalizar los hilos de Player, Objeto (si existiera), Disparo y el hilo principal.

Clase:	GameViewTour	Herencia:	SurfaceView
Miembros:	MiThreadTour _miThread int STATE_LOSE, STATE_PAUSE, STATE_READY int STATE_RUNNING, STATE_WIN, STATE_SLEEPED int modo_juego, int ESTADO_VICTORIA, ESTADO_MUERTE_DERECHA int ESTADO_MUERTE_IZQUIERDA, SIN_ESTADO int modoDisparo int SIN_DISPARO, DISPARO_PAUSA int mMode Paint paintSuperior, textoNaranja, textoAmarillo, textoNaranjaBig		

	Paint textoVerde, rojo, fondoNegro, borde Personaje player MediaPlayer gameover, victoria, perderVida Drawable fondo Bitmap vidas, vidas2, armas, disparoPausa int puntuación, numVidas, time Rect rectSuperior, rectInferior int width, height, staticHeight, acelerometro Context miContexto Handler handler Disparo disparo Canvas canvas Objeto objetoDisparoPausa, vidaExtra, fruta ArrayList<Pelota> _arrayPelotas, _arrayAuxiliar ArrayList<BloqueRompible> _arrayBloques ArrayList<BloqueIrrompible> _arrayBloquesPared boolean colisionDisparoBloquePared, colisionPelotaBloque boolean procesoCarga, musicaFinal, ready long tiempoInicio int tiempoPredefinido
Métodos:	void surfaceCreated(SurfaceHolder holder) void surfaceChanged(SurfaceHolder holder, int format, int width, int height) void surfaceDestroyed(SurfaceHolder holder) void finThreadDisparo() void finalizarThreads() void inicializarVariables() boolean onTouchEvent(MotionEvent event) void onDraw(Canvas canvas) void actualizarPosiciones() void colisionesBloquePelota(Bloque bloque, Pelota pelota) void explosionPelota(Pelota pelota, Canvas canvas, Iterator<Pelota> iterador) void cancelarObjeto() void eventoVida() void victoria() void gameOver() void vidaPerdida() void setState(int state), int getState() void onSensorChanged(SensorEvent event)

Tabla 3.64. Clase GameViewTour. Miembros y métodos.

La clase **MiThreadTour** (tabla 3.65) es una clase interna de GameViewTour y representa el hilo principal de una partida en modo Tour. Permanece activa desde que se carga la partida por primera vez hasta que se acaba la partida o el usuario la abandona.

Tiene como miembros un objeto SurfaceHolder, que permite controlar el SurfaceView, un objeto GameViewTour y un boolean que controla la ejecución del hilo.

Al iniciarse el Thread (llamada start() sobre el objeto de la clase) se llama automáticamente al método run(). Este método se encarga de, una vez sincronizado el objeto SurfaceHolder, llamar al método onDraw() de GameViewTour para pintar toda la pantalla, actualizar la posición del personaje llamando a update() sobre el objeto Personaje y llamar al método actualizarPosiciones() de GameViewTour, que actualiza las posiciones de todas las pelotas y bloques de la pantalla. Una vez realizadas estas acciones, ejecuta un simple cálculo para asegurar un framerate constante de la aplicación, limitándolo a un máximo de 50 fps. Por último, se encarga de finalizar la partida si el tiempo llega a 0, en cuyo caso llama al método eventoVida() de la clase GameViewTour.

Dicho método run() depende del estado del juego. Si el estado es distinto a Running, solamente realiza la llamada a onDraw() de manera que no se actualiza ni la posición del personaje ni la de las pelotas.

Clase:	MiThreadTour	Herencia:	Thread
Miembros:	SurfaceHolder _surfaceHolder GameViewTour _gameViewTour boolean _run		
Métodos:	SurfaceHolder getSurfaceHolder() void setRunning(boolean run); boolean getRunning() void run()		

Tabla 3.65. Clase MiThreadTour. Miembros y métodos.

La clase **GamePanic** (tabla 3.66) es muy similar a la clase GameTour en cuanto a funcionamiento. Se ha decidido utilizar dos clases diferentes para los dos modos de juego diferentes ya que, a pesar de contar con casi los mismos miembros y métodos, existen varias diferencias en su funcionamiento.

Solo hay dos miembros diferentes respecto a GameTour. Un int que representa el número del nivel de la partida, y un objeto GameViewPanic (en lugar del objeto GameViewTour).

En cuanto a métodos, ambas clases implementan los mismos métodos pero con distinto funcionamiento. En onCreate(), los archivos de máximas puntuaciones creados en ambos casos son diferentes. Además la implementación de los objetos Handler también son diferentes en ambos casos, ya que hay diferencias entre ambas modalidades del juego. También el registro del sensor acelerómetro es diferente por necesitar registrarse en clases distintas.

Clase:	GamePanic	Herencia:	Activity
Miembros:	boolean sonar String modoDeJuego Intent i GameViewPanic gameViewPanic SensorManager sensores Sensor acelerometro int level AudioManager am		
Métodos:	void onCreate(Bundle savedInstanceState) void onResume() void onPause() void onDestroy() void crearDialogo() boolean onKeyDown(int keyCode, KeyEvent event)		

Tabla 3.66. Clase GamePanic. Miembros y métodos.

La clase **GameViewPanic** (tabla 3.67) hereda de SurfaceView e implementa el interfaz SurfaceHolder.Callback de igual manera que GameViewTour. Tiene un comportamiento similar a dicha clase en líneas generales pero cuenta con varias diferencias. Al igual que entre GamePanic y GameTour, se ha decidido diferenciar entre GameViewPanic y GameViewTour debido a estas diferencias.

Visualmente la única diferencia es la eliminación del tiempo de la partida por una barra que se incrementa cuando el personaje destruye o rompe alguna pelota. Cuando esta barra está llena y el personaje rompe una bola, se reinicia el contador de la barra y se incrementa en uno el nivel de la partida.

En cuanto a miembros existen varias diferencias. Se sustituye el nivel de la pantalla por un entero que representa el nivel mostrado en la barra de nivel. También aparece un valor entero que mide tiempo, aunque en este caso el valor temporal es usado para medir el tiempo de caída entre pelotas en lugar de medir el tiempo de la pantalla. Se añade un boolean para identificar un tipo de pelota especial de este modo de juego, y un array que almacena los diferentes bitmaps mostrados en la barra de nivel, sustituyendo al array de bloques ya que en esta modalidad los bloques no aparecen. Otra diferencia importante es la clase que maneja el hilo principal de la partida, en este caso se trata de la clase MiThreadPanic.

Respecto a los métodos, se implementan los mismos métodos con una salvedad, en GameTour se utiliza el método colisionPelotaBloque(), el cual en modo Panic no se necesita y se incluye el método caidaPelota() que crea una pelota nueva y la incorpora al array de pelotas de la partida.

A pesar de que casi todos los métodos implementados son los mismos, su funcionalidad no es equivalente. En el constructor se cargan las variables que no se van a modificar durante la partida como los diferentes fondos, bitmaps, objetos MediaPlayer para los eventos de partida, y los efectos de sonido (en este caso solo uno, la explosión de las pelotas). El comportamiento de los métodos surfaceChanged(), surfaceCreated() y surfaceDestroyed() no sufre variación. En caso de inicio de partida se recurre al método inicializarVariables() que crea los Paint necesarios para configurar los textos, el objeto de la clase Personaje y una sola instancia de la clase Pelota que caerá inicialmente.

Los estados de partida son similares a GameViewTour, de manera que inicialmente la partida se encuentra en estado Ready. El usuario debe pulsar la pantalla para pasar a estado Running y debe volver a pulsar para realizar un disparo (eventos detectados en onTouchEvent()). Al pulsar la tecla back se muestra el cuadro de dialogo que permite al usuario abandonar la aplicación (método crearDialogo() en GamePanic) entrando la aplicación en estado Pause. Al igual que en GameViewTour, cuando la aplicación deja de estar visible, el SurfaceView se destruye y se pasa al estado Sleeped que mantiene en funcionamiento la aplicación, sin mostrar nada.

El método onDraw() se encarga de pintar el fondo y borde (en función de nivel de la pantalla), los textos y bitmaps de las vidas y niveles, el personaje y las pelotas de la partida. Durante los procesos de carga de pantalla solo muestra una pantalla en negro y cuando ocurre un evento de victoria o vida perdida se encarga de llamar a los métodos victoria() o eventoVida(). También se encarga de actualizar una variable utilizada para medir el tiempo activo de la partida y llama al método caidaPelota() cuando dicho tiempo cumple un intervalo que depende del nivel de la partida.

Dicho método solamente crea una pelota (azul o roja) que se añade al array de pelotas. Comienza cayendo desde el techo de la pantalla si el numero de pelotas no es muy elevado y crea una pelota especial, que sirve para romper todas las demás pelotas en un instante, en caso de haber un numero de pelotas muy elevado en pantalla, lo cual ayuda mucho al jugador.

El método actualizarPosiciones() actualiza las posiciones de las pelotas y detecta las colisiones entre pelota-disparo y pelota-player. La primera de las colisiones llama al método explosionPelota() que destruye la pelota colisionada y crea dos nuevas de un tamaño inferior aumentando la puntuación de la partida. El segundo tipo de colisión hace al personaje perder una vida, cargando el método eventoVida() quien decide si la perdida de vida conlleva el fin de la partida o no, y se lo comunica a GamePanic mediante el objeto Handler.

Con el método onSensorChanged() se detecta el sensor acelerómetro y en función de su valor se establece la dirección de movimiento del personaje y establece el modo

de animación del mismo. Por último el manejo de los estados de la partida se realiza mediante los métodos getState() y setState(int state).

La clase **MiThreadPanic** (tabla 3.68) es una clase interna a GameViewPanic y su funcionamiento es exactamente igual que MiThreadTour respecto a GameViewTour, sustituyendo el objeto GameViewTour por un objeto GameViewPanic.

La clase **MusicaDeFondo** (tabla 3.69) hereda de Service y se ejecuta en segundo plano, de manera que el usuario no puede interactuar con dicho servicio directamente. Es creado por GamePanic o GameTour y permanece activo hasta la llamada a stopService(Intent i).

Cuenta como miembros con un objeto MediaPlayer y el contexto de la actividad que la llama.

En el método onStart(), crea un MediaPlayer que dependerá de si el usuario ha elegido el modo Tour o modo Panic y dentro del modo Tour dependerá de la pantalla. El método finDeTiempo() se ejecuta en el modo Tour, y cambia la música del servicio cuando queda poco tiempo para finalizar la partida a modo de aviso al jugador. El método finDeTiempoPanic() se ejecuta en modo Panic y cambia la música del servicio cuando aparece la pelota especial en pantalla, volviendo a la música normal cuando dicha pelota ha desaparecido llamando al método normalidadPanic(). Todas las melodías de pantalla se configuran en bucle, de manera que cuando finaliza la canción se vuelve a reproducir.

Los métodos onStop(), arrancaPlayer() y pause() se encargan de parar, iniciar o pausar el MediaPlayer si la preferencia de sonido esta activada. El método onDestroy() para el MediaPlayer.

Clase:	GameViewPanic	Herencia:	SurfaceView.
Miembros:	MiThreadPanic _miThread int STATE_LOSE, STATE_PAUSE, STATE_READY int STATE_RUNNING, STATE_WIN, STATE_SLEEPED int modo_juego int ESTADO_VICTORIA, ESTADO_MUERTE_DERECHA int ESTADO_MUERTE_IZQUIERDA, SIN_ESTADO int modoDisparo, mMode, SIN_DISPARO, DISPARO_PAUSA Paint paintSuperior, textoNaranja, textoAmarillo, textoNaranjaBig Paint textoVerde, fondoNegro, borde Personaje player MediaPlayer gameover, victoria, perderVida Boolean explosionEstrella ArrayList<Drawable> fondos ArrayList<Paint> bordes Drawable fondos Bitmap vidas, vidas2 int puntuación, numVidas, width, height, staticHeight, acelerometro Rect rectSuperior, rectInferior Context miContexto Handler handler Disparo disparo Canvas canvas ArrayList<Pelota> _arrayPelotas, _arrayAuxiliar ArrayList<Bitmap> _arrayLevel int tiempoEntrePelotas, time, tiempoPredefinido boolean procesoCarga, caidaPelota, ready long tiempoInicio		
Métodos:	void surfaceCreated(SurfaceHolder holder) void surfaceChanged(SurfaceHolder holder, int format, int width, int height) void surfaceDestroyed(SurfaceHolder holder) void finThreadDisparo() void finalizarThreads() void inicializarVariables() boolean onTouchEvent(MotionEvent event) void onDraw(Canvas canvas) void actualizarPosiciones() void explosionPelota(Pelota pelota, Canvas canvas, Iterator<Pelota> iterador) void eventoVida() void victoria() void gameOver() void vidaPerdida() void setState(int state), int getState() void onSensorChanged(SensorEvent event)		

Tabla 3.67. Clase GameViewPanic. Miembros y métodos.

Clase:	MiThreadPanic	Herencia:	Thread
Miembros:	SurfaceHolder _surfaceHolder GameViewPanic _gameViewPanic boolean _run		
Métodos:	SurfaceHolder getSurfaceHolder() void setRunning(boolean run), boolean getRunning() void run()		

Tabla 3.68. Clase MiThreadPanic. Miembros y métodos.

Clase:	MusicaDeFondo	Herencia:	Service
Miembros:	MediaPlayer mp Context contexto		
Métodos:	void onStart(Intent intent, int startId) MediaPlayer getMediaPlayer() void finDeTiempo() void finDeTiempoPanic() void normalidadPanic() void onStop() void arrancaPlayer() void pause() void onDestroy()		

Tabla 3.69. Clase MusicaDeFondo. Miembros y métodos.

La clase **Pelota** (tabla 3.70) representa las pelotas que el jugador debe destruir. Tiene como miembros el bitmap que se pinta en pantalla, un objeto Coordenadas, un Rect para la detección de colisiones, variables relacionadas con la posición, velocidad y dirección de la pelota y un String para identificar el tamaño de cada pelota. En el juego aparecen pelotas de cuatro tamaños diferentes (grande, media, pequeña y enana) y dos colores diferentes (azul y rojo) además de una pelota exclusiva del modo Panic denominada estrella, mostradas en la figura 3.18.

Figura 3.18. Pelotas. Tamaños y colores.

Como métodos cuenta con toogleXDirection() que sirve para cambiar la dirección en el eje X. El método nuevaPelota() es usado para crear una pelota adicional asignán-

dola una posición, velocidad, tamaño y dirección en el eje X. El método update() se encarga de actualizar la posición y velocidad de los ejes X e Y, así como las coordenadas del rectángulo usado para detectar colisiones.

Además se compone de de sets y gets para acceder a las variables de posición, velocidad, tamaño, rectángulo de colisión, bitmap y objeto Coordenadas.

Clase:	Pelota	Herencia:	
Miembros:	Bitmap _bitmap Coordenadas _coordenadas Rect rectanguloColisionPelota double vy0, y, vx0, a int x, x0 int bitmapId, ancho double velocidadMaximaEnana, velocidadMaximaPequena double velocidadMaximaMedia, velocidadMaximaGrande String tamaño int DIRECCION_X_DERECHA, DIRECCION_X_IZQUIERDA int _direccionX		
Métodos:	void setVelocidadY(double speed); double getVelocidadY() void setVelocidadX(double speed); double getVelocidadX() double getVelocidadMaximaEnana() double getVelocidadMaximaPequeña() double getVelocidadMaximaMedia() double getVelocidadMaximaGrande() int getAncho(); void setAncho(int ancho) int getXDirection(); void setXDirection(int direccion) void toggleXDirection() void nuevaPelota(double velocidad, int id, String tamano, int direccionX) void setA(double valor_a) void setY(int a); int getY() void update() Rect getRectanguloColision() void setRectanguloColision(int left, int top, int right, int bottom) int getBitmapId(); void setBitmapId(int id) void setGraphic(Bitmap bitmap); void setTamano(String tamano) String getTamano() Bitmap getGraphic() Coordenadas getCoordenadas()		

Tabla 3.70. Clase Pelota. Miembros y métodos.

La clase **Coordenadas** (tabla 3.71) es una clase interna de Pelota y se utiliza para tener constancia de las posiciones en los ejes X e Y de cada pelota. Únicamente cuenta con dos miembros que almacenan estos dos valores.

Clase:	Coordenadas	Herencia:	
Miembros:	int _xPelota int _yPelota		
Métodos:	int getX(); void setX(int value) int getY(); void setY(int value)		

Tabla 3.71. Clase Coordenadas. Miembros y métodos.

La clase **Personaje** (tabla 3.72) representa al personaje que el jugador mueve por la pantalla mediante los acelerómetros.

Sus miembros son el contexto de la actividad, un String usado en la clase Disparo para identificar el modo de juego, un bitmap, un objeto Thread utilizado para la animación de desplazamiento del personaje, variables relacionadas con el tamaño y posición del bitmap, un objeto Rect para la detección de colisiones, variables boolean utilizadas en la animación del personaje y variables int que representan si el personaje se mueve a izquierda, derecha o está parado.

En el constructor se le asigna un bitmap inicial, se crea el rectángulo para la detección de colisión y se inicia el Thread de la clase llamando al método hiloAnimacion(). Este hilo se encarga de asignar al personaje una serie de bitmaps consecutivos a modo de animación, según la variable modoAnimacion. Posee tres métodos, cambiaVictoria(), cambiaMuerteDerecha() y cambiaMuerteIzquierda(), que modifican el bitmap del personaje en caso de obtener una victoria, recibir el impacto de una pelota por la derecha o recibirlo por la izquierda. En la figura 3.19 se muestran algunos de los bitmaps que aparecen durante el juego.

a) Bitmap inicial b) Bitmap victoria c) Bitmap muerte

Figura 3.19. Personaje. Bitmaps.

Se tiene un método, runAnimacion(boolean run), que controla la duración del hilo de la clase. Además, mediante el método finThread(), se interrumpe automáticamente el hilo. Este método es llamado cuando el usuario elige abandonar la partida voluntariamente.

Se cuenta también con un método, setPosition(), que establece la posición del personaje y es llamado cuando se crea el objeto a fin de colocarlo inicialmente en una posición centrada de la pantalla. Mediante el método update() se actualiza la posición del personaje y el rectángulo de colisión.

Clase:	Personaje	Herencia:	
Miembros:	Context context String modo Bitmap personaje int posX, posY, int width Thread t Rect rectanguloColisionPlayer int ANIMACION_DERECHA, ANIMACION_IZQUIERDA int NO_ANIMACION, int modoAnimacion boolean isRunning, boolean primeraVez		
Métodos:	Bitmap getGraphic Rect getRectanguloColision void setRectanguloColision(int left, int top, int right, int bottom) String getModo(), void setModo(String modo) void cambiaVictoria(Context context) void cambiaMuerteDerecha(Context context) void cambiaMuerteIzquierda(Context context) void runAnimacion(boolean running) void finThread() void hiloAnimacion() void setPosition(int pos_x, int pos_y, int width, int alto) void update(int acelerometro) int getPosicionX ();void setPosicionX(int x) int getPosicionY ();void setPosicionY(int y)		

Tabla 3.72. Clase Personaje. Miembros y métodos.

La clase **Objeto** (tabla 3.73), representa los diferentes ítems que el personaje puede coger durante la partida. Puede ser de tres tipos (figura 3.20): vida extra (añade una vida al número total de vidas), fruta (aumenta la puntuación total) y disparo pausa (proporciona al personaje un disparo diferente del inicial). Si el personaje rompe un bloque que contiene un Objeto, éste cae al suelo y durante cinco segundos puede ser cogido. Si pasados cinco segundos, el personaje no ha obtenido el Objeto, éste desaparece.

a) Disparo pausa b) Fruta c) Vida extra

Figura 3.20. Objetos. Tipos.

Tiene como miembros un bitmap, variables para manejar la posición del objeto, un Rect para detectar colisiones, un objeto Thread, el canvas sobre el que se pinta (el canvas de la clase que instancia Objeto) y un boolean que controla el método run del hilo.

Inicialmente, en el constructor, se asigna a la instancia un bitmap, dependiendo del tipo de objeto, las posiciones X e Y del bloque destruido desde las que parte el Objeto, se crea el Rect para detectar las colisiones y se llama al método hiloObjeto(). Éste método crea el Thread que maneja la vida del Objeto y lo inicia, de manera que su posición se actualiza constantemente hasta llegar al suelo, se calcula el tiempo del sistema y si cinco segundos después, el personaje no ha capturado el ítem, desparece. Si lo captura antes de esos cinco segundos, igualmente desaparece en el momento de ser capturado.

También posee un método, pintarObjeto(), que pinta el objeto en el canvas con las posiciones actualizadas y diferentes métodos get y set.

Clase:	Objeto	Herencia:	
Miembros:	Bitmap bitmap int x, y boolean existeObjeto Thread t Canvas canvas Rect rectanguloObjeto		
Métodos:	Bitmap getGraphic() void setPosicionX(int x); int getPosicionX() void setPosicionY(int y); int getPosicionY() void setObjetoRunning(boolean running_existe) boolean getObjetoRunning() void pintarObjeto(Canvas canvas, int x, int y) void setRectanguloColision(int left, int top, int right, int bottom) Rect getRectanguloColision() void hiloObjeto()		

Tabla 3.73. Clase Objeto. Miembros y métodos.

La clase **Disparo** (tabla 3.74) representa los disparos que se producen cuando el usuario, durante la partida, pulsa la pantalla. Un objeto Disparo está representado visualmente por dos bitmaps, la punta del disparo y su tronco.

Tiene como miembros los citados bitmaps, un int y un boolean para identificar las situaciones en las que el disparo estándar se sustituye por otro tipo de disparo, el contexto de la clase que llama al objeto, un Thread que se encarga de controlar el disparo,

variables relacionadas con la posición del disparo, el Canvas sobre el que se pinta, un Rect para detectar colisiones, el objeto Personaje utilizado en la partida ya que el disparo se inicia en la posición de dicha instancia y varias variables boolean e int utilizadas en el Thread relacionadas con el disparo.

En el constructor se asigna al objeto los diferentes bitmaps, se crea el Rect para detectar las colisiones y se inicia el Thread llamando a hiloDisparo().

Este método crea el objeto Thread y arranca su método run(), el cual estará funcionando hasta la finalización del nivel o pantalla. Una vez que el usuario pierda una vida o gane la partida, el objeto Disparo se destruye, creándose de nuevo en la siguiente pantalla que se juegue. Cada vez que se realice un disparo, a su finalización, se mantiene el hilo en espera utilizando wait() sobre el hilo y cuando el usuario decide volver a disparar, se activa el hilo nuevamente utilizando notifyAll() (llamando al método activarDisparo()). Sin estas dos llamadas, el disparo se realizaría constantemente sin ningún tipo de pausa y no es lo que se busca.

El disparo se inicia en la coordenada X del personaje a nivel del suelo y progresivamente va aumentando su coordenada y hasta llegar al techo donde finaliza, salvo que colisione antes con alguna pelota (en cuyo caso se destruye la pelota y se finaliza el disparo). Se dibuja la punta del disparo y posteriormente, en función de la distancia que hay entre la punta del disparo y el suelo de la partida, se dibuja repetidamente el bitmap del tronco del disparo para dar la sensación de unión entre los diferentes bitmaps. En cada iteración del hilo, a la vez que se dibuja el disparo, se actualiza su rectángulo de colisión.

Hay dos tipos de disparo (figura 3.21), el disparo estándar con una punta semejante a una lanza y un tronco curvilíneo, y el disparo pausa, con una punta similar a una especie de gancho y un tronco lineal. El primer tipo finaliza nada mas impactar con el techo o una pelota. El segundo tipo tiene la peculiaridad que cuando se encuentra con un bloque que no se puede romper o con el techo, permanece estático durante unos instantes. Finaliza cuando una pelota golpea al disparo, cuando el disparo golpea un bloque rompible o cuando se acaba el tiempo en el cual el disparo permanece estático.

a) Disparo estándar b) Disparo pausa

Figura 3.21. Disparos. Tipos

Además de los métodos citados, y de los diferentes gets y sets, se utilizan posicionDisparo() para pintar el Disparo sobre el Canvas, el método cambiaBitmaps() que detecta si se pasa del disparo estándar al disparo pausa para posteriormente cambiar los bitmaps correspondientes y el método finDisparo() llamado cuando un disparo

golpea una pelota, un bloque o llega al techo y en el cual se reinicia la posición Y del objeto Disparo antes de pausar el hilo principal para que durante la próxima iteración comience el Disparo a la altura del suelo.

Clase:	Disparo	Herencia:	
Miembros:	Bitmap disparo, paloDisparo int tipoDisparo boolean disparoPausa Context contexto Thread t int posicionX, posicionY boolean isRunning, isShooting, finDisparo Canvas canvas int SI_DISPARO, SIN_DISPARO, DISPARO_PAUSA int modoDisparo Rect rectanguloColisionDisparo boolean mantenerBloque, colisionDisparo int tiempoMantener, contadorTiempo Personaje personaje		
Métodos:	Bitmap getGraphic() boolean estaDisparando() Rect getRectanguloColision() void setRectanguloColision(int left, int top, int right, int bottom) void setYDisparo(int y); int getYDisparo() void activarDisparo(int id, int posicion) void posicionDisparo(Canvas canvas) void cambiaBitmaps(int tipo) void setRunning(boolean running); boolean getRunning() void setColisionDisparo(boolean colision) boolean getColisionDisparo() Thread getThread() void finDisparo() void hiloDisparo()		

Tabla 3.74. Clase Disparo. Miembros y métodos.

La clase **Efectos** (tabla 3.75) se usa para reproducir los efectos de sonido durante el juego tales como la explosión de una pelota, la destrucción de un bloque o la captura de un ítem.

Tiene como miembros un objeto SoundPool que reproduce los efectos, un objeto HashMap para almacenar los diferentes objetos, un objeto AudioManager para indicarle

al SoundPool que tipo de sonido debe reproducir (en este caso el sonido multimedia) y el contexto de la actividad en cuestión.

Se ha utilizado SoundPool, en lugar de MediaPlayer, debido a que es más ligero y funciona mejor con archivos de pequeño tamaño ya que no permite archivos muy grandes (razón por la cual no se ha utilizado para la música de fondo).

En el constructor se inicializan los cuatro miembros y en el método addEfecto(int index, int soundID) se almacena en la posición index, el sonido al que se accede con el identificador soundID dentro del HashMap para su posterior lectura. A este método se le llama una vez por cada efecto almacenado. El método playEfecto() se utiliza para reproducir el sonido indicado.

Por último, antes de abandonar la aplicación se hace una llamada a cleanup() que sirve para limpiar el HashMap.

Clase:	Efectos	Herencia:	
Miembros:	SoundPool mSoundPool; HashMap<Integer,Integer> mSoundPoolMap; AudioManager am; Context context;		
Métodos:	void initEfectos(Context myContext) void addEfecto(int index,int soundID) void playEfecto(int index,float speed) void cleanup()		

Tabla 3.75. Clase Efectos. Miembros y métodos.

La clase **Bloque** (tabla 3.76) es una clase abstracta que sirve para crear dos tipos de bloques: BloqueRompible y BloqueIrrompible.

Tiene como miembros los atributos comunes a ambas clases, un Rect para detectar colisiones, las posiciones X e Y del bloque, un boolean que identifica si el bloque se coloca horizontal o verticalmente y varias variables int que sirven para asignar un objeto Objeto al bloque.

En esta clase solamente se implementan los métodos get y set para los miembros anteriores, además de los métodos abstractos getGraphic(), setGraphic(), escalarBitmap() y pintarBloque(), implementados en BloqueRompible y BloqueIrrompible.

Clase:	Bloque	Herencia:	
Miembros:	Rect rectanguloColisionBloque int posicionX, posicionY, icono boolean rotar int VIDA_EXTRA, DISPARO_PAUSA,FRUTA		
Métodos:	void setRotar(boolean rotar); boolean getRotar() void setX(int x); int getX() void setY(int y); int getY() Rect getRectanguloColision() void setRectanguloColision(int left, int top, int right, int bottom) void setIcono(int tipo); int getIcono() Bitmap getGraphic(); void setGraphic(Bitmap bitmap) escalarBitmap(int anchura, int altura, boolean rotar) pintarBloque(Canvas canvas, int left, int top)		

Tabla 3.76. Clase Bloque. Miembros y métodos.

Las clases **BloqueRompible** y **BloqueIrrompible** (tablas 3.77 y 3.78) tienen los mismos miembros y métodos. La diferencia únicamente reside en el tipo de bitmap utilizado (naranja en BloqueRompible y amarillo en BloqueIrrompible) y en su posibilidad de destrucción (evaluado en GameViewTour). Su único miembro es un bitmap que representa la imagen que ve el usuario en pantalla. Los dos tipos de bitmap se muestran a continuación (figura 3.22):


a) Bloque rompible b) Bloque irrompible

Figura 3.22. Bloques. Tipos.

Los métodos que implementan son los métodos abstractos de la clase padre getGraphic() y setGraphic() para obtener el bitmap, escalarBitmap() que sirve para crear un bitmap escalado e indica si el bloque se coloca en posición horizontal o vertical, y pintarBloque() que simplemente pinta el bloque sobre el canvas indicado, en las posiciones indicadas.

Clase:	BloqueRompible	Herencia:	Bloque
Miembros:	Bitmap bitmapBloque		
Métodos:	Bitmap getGraphic(), void setGraphic(Bitmap bitmap) Bitmap escalarBitmap(int anchura, int altura, boolean rotar) void pintarBloque(Canvas canvas, int left, int top)		

Tabla 3.77. Clase BloqueRompible. Miembros y métodos.

Clase:	BloqueIrrompible	Herencia:	Bloque
Miembros:	Bitmap bitmapBloque		
Métodos:	Bitmap getGraphic(), void setGraphic(Bitmap bitmap) Bitmap escalarBitmap(int anchura, int altura, boolean rotar) void pintarBloque(Canvas canvas, int left, int top)		

Tabla 3.78. Clase BloqueIrrompible. Miembros y métodos.

Las clases **RankingTour** y **RankingPanic** (tablas 3.79 y 3.80) muestran la pantalla de máximas puntuaciones para el modo Tour y modo Panic respectivamente, compuesto por un archivo xml que se establece como vista de la actividad y un fichero de texto con las puntuaciones. Ambas clases son idénticas en funcionamiento pero difieren en dos de sus miembros.

En RankingTour el archivo de texto utilizado es ranking_tour.txt y en RankingPanic se utiliza ranking_panic.txt, mientras que respecto al archivo xml, la clase RankingTour accede a ranking_tour.xml y la clase RankingPanic accede a ranking_panic.xml.

Los miembros de la clase son un String que representa el fichero de texto, dos variables int que representan la pantalla y la puntuación de la partida actual, otro String que representa el identificador de puntuación que el usuario ha escogido (en caso de ser necesario), arrays de TextView para almacenar puntuaciones, identificadores de puntuación y números de pantalla, un array de String para leer del fichero y posteriormente almacenar los datos en los arrays de TextView y un identificador de actividad.

En el método onCreate() se carga la vista definida en el archivo xml, y se configura mediante el método inicializar(). Se lee el fichero de texto almacenando cada línea en el array de String y cada posición del array se divide en puntuaciones, nombres y pantallas, cuyos datos son almacenados en los array de TextView correspondientes. Se comprueba la puntuación de la partida y si dicha puntuación se encuentra entre las cinco mejores, se le pide al usuario introducir un nombre para posteriormente actualizar los arrays y añadir la nueva puntuación en el método reorganizarPuntuaciones(), en el cual también se actualiza el fichero de texto. Si la puntuación de la partida no se encuentra entre las mejores, simplemente se muestra la vista. En ambos casos, cuando el usuario toque la pantalla, se cierra la actividad y se vuelve al menú principal, mediante el método onTouchEvent().

En el momento de introducir nombre, se carga la clase PuntuacionDialog mediante startActivityForResult(Intent j, int ID). Este método se utiliza cuando se necesita obtener una respuesta al finalizar la clase, en este caso PuntuacionDialog, quien debe devolver el nombre introducido por el usuario. El parámetro ID representa la clase que realiza la llamada. Para recibir el valor devuelto de PuntuacionDialog se implementa el método

onActivityResult(), quien simplemente captura el nombre introducido y llama a reorganizarPuntuaciones().

Clase:	RankingTour	Herencia:	
Miembros:	String rankingTour int puntuacion, pantalla String nombreInsertado TextView[] arrayPuntuaciones, arrayNombres, arrayPantallas String [] arrayFinal int ID		
Métodos:	void onCreate(Bundle savedInstanceState) void onActivityResult(int requestCode, int resultCode,Intent data) boolean onTouchEvent(MotionEvent event) void inicializar() void reorganizarPuntuaciones(int puntuacion, String nombre, int pantalla)		

Tabla 3.79. Clase RankingTour. Miembros y métodos.

Clase:	RankingPanic	Herencia:	
Miembros:	String rankingTour int puntuacion, pantalla String nombreInsertado TextView[] arrayPuntuaciones, arrayNombres, arrayPantallas String [] arrayFinal int ID		
Métodos:	void onCreate(Bundle savedInstanceState) void onActivityResult(int requestCode, int resultCode,Intent data) boolean onTouchEvent(MotionEvent event) void inicializar() void reorganizarPuntuaciones(int puntuacion, String nombre, int pantalla)		

Tabla 3.80. Clase RankingPanic. Miembros y métodos.

La clase **PuntuacionDialog** (tabla 3.81) se encarga de pedir un identificador de puntuación al usuario tras haber conseguido una de las máximas puntuaciones. Su único miembro es un String que almacena el valor del nombre elegido por el usuario.

En onCreate() se carga la vista mostrada en la actividad mediante el fichero set_puntuacion.xml. Se obtienen el EditText y Button de dicho fichero y se implementan sus escuchadores de manera que cuando se pulse el botón tras haber introducido el nombre, se envía el nombre a la clase llamadora mediante el método setResult(int ID, Intent i), añadiendo el nombre al objeto Intent.

También se implementa el método onKeyDown() para impedir que se cierre la actividad pulsando back y de esta forma forzar al usuario a introducir nombre.

Clase:	PuntuacionDialog	Herencia:	
Miembros:	String nombre		
Métodos:	void onCreate(Bundle savedInstanceState) boolean onKeyDown(int keyCode, KeyEvent event)		

Tabla 3.81. Clase PuntuacionDialog. Miembros y métodos

3.3. Implementación

Esta sección explica detalladamente los aspectos relevantes e importantes para la implementación de la aplicación. Sirve como ayuda a este apartado el anterior estudio de análisis y diseño de la aplicación. La implementación se ha realizado siguiendo un incremento funcional progresivo, esto es, comenzar desde la funcionalidad más básica como es el paso entre actividades o la carga de archivos xml y progresivamente ir incrementando la funcionalidad, permitiendo en todo momento que la aplicación sea operativa.

Este capítulo se estructura de manera similar al desarrollo del juego: comienza con la creación de actividades y como navegar entre ellas o la carga de interfaces de usuario en las distintas actividades y posteriormente se desarrolla el núcleo principal del juego. El último paso es añadir detalles como música, efectos de sonido o diferentes acciones en caso de conseguir una victoria o perder una vida.

3.3.1. Implementación previa y cambios realizados

Antes de comenzar con la implementación de la aplicación, se ha considerado necesario la realización de tutoriales a modo de aprendizaje sobre la plataforma Android para familiarizarse con la creación de actividades y la carga de elementos xml [19].

Tras familiarizarse con los elementos más básicos de la plataforma, se analizó el código de ejemplo de la web oficial de desarrolladores Android del juego LunarLander [20] que sirvió como ayuda para el planteamiento inicial de la aplicación.

Durante la implementación de la aplicación se han realizado cambios respecto a la idea inicial que se tenía de algunos aspectos del juego. Inicialmente se pensó en dar un movimiento lineal a los objetos de la clase Pelota, pero rápidamente se desestimó dicha

opción ya que, a pesar de ser mucho más fácil de implementar, no ofrecía una buena sensación de juego y se sustituyo por un movimiento con trayectoria parabólica que proporciona al usuario una jugabilidad mas adictiva y divertida.

Otro elemento que se modificó respecto a la idea inicial fue la orientación de las pantallas. Inicialmente se pensó en permitir a la aplicación la posibilidad de mostrarse en orientación vertical u horizontal de manera que cada usuario eligiera la opción que mejor se adaptase a sus gustos. Sin embargo esta idea se mostró visualmente ineficiente en momentos en los que la pantalla se llena de pelotas y la orientación era vertical ya que dejaba poco margen de maniobra al personaje. Por este motivo se fuerza a las distintas actividades a mostrarse únicamente en horizontal. Esto se consigue incluyendo dentro de cada actividad declarada en el fichero AndroidManifest.xml la sentencia indicada en la figura 3.23.

```
android:screenOrientation="landscape"
```

Figura 3.23. Orientación horizontal.

Una vez que el juego se encontró en una fase de desarrollo avanzada, se decidió modificar el ciclo de vida del hilo principal de la partida. Debido al funcionamiento de la clase SurfaceView, al crearse, llama automáticamente a los métodos surfaceCreated() y surfaceChanged(), y el SurfaceView permanece activo mientras se muestra en pantalla. Si durante la ejecución de la partida se sale al escritorio o se recibe una llamada por ejemplo, deja de estar visible y el SurfaceView se destruye automáticamente (pero la clase que lo implementa y todos sus métodos y variables continúan activos), volviéndose a crear nuevamente cuando se vuelve a mostrar en pantalla.

Siguiendo el ejemplo LunarLander mencionado anteriormente, el hilo principal de la partida se creaba en el citado método surfaceCreated() y se finalizaba en surfaceDestroyed(), de manera que si el usuario salía al escritorio aparecían problemas con los hilos de la aplicación. Para solucionar esto, es el constructor de la clase quien inicia el hilo de la partida de manera que mientras la clase este activa (mostrando o no el SurfaceView), el hilo permanece activo y solamente es finalizado cuando se cierra completamente la clase.

3.3.2. Interacción entre actividades

Antes de centrarse en la navegación entre actividades se muestra una breve introducción acerca de las actividades y las diferentes formas de navegar entre ellas implementadas en el juego.

Una actividad es una clase que hereda de Activity en la cual es necesario implementar los métodos de devolución a los que el sistema llama durante los diferentes estados del ciclo de vida de la actividad (creado, parado, reanudado o destruido). Los dos métodos más importantes son onCreate() y onPause().

El método onCreate() es de implementación obligatoria y se recomienda inicializar en él los componentes esenciales de la actividad. Lo más importantes es llamar a setContentView() para asignar una vista a la actividad. El método onPause() es llamado por el sistema como primer indicador de que el usuario está abandonando la actividad, aunque esto no quiere decir que se deba destruir.

Para poder ejecutar cualquier actividad, es imprescindible incluirla en el fichero AndroidManifest.xml, como se indica en la figura 3.24.

```
<activity android:name=".NombreDeLaClase" />
```

Figura 3.24. Incluir actividad en el fichero manifiesto.

Una vez creadas las actividades, el siguiente paso fue navegar a través de ellas. El objetivo era ir de una actividad a otra a través de botones o imágenes que permitían pulsar sobre ellos.

La clase SuperPang llama a MenuPang mediante la pulsación de un botón. Para ello, dentro del método que implementa el escuchador del botón se incluye el código de la figura 3.25. En dicho código se crea un objeto Intent, el cual es una descripción abstracta de una operación a realizar, indicando el contexto de la aplicación y la clase que se va a ejecutar, y posteriormente se llama a startActivity() quien se encarga de cargar la nueva actividad. Aunque hay otras formas de construir un objeto Intent, se ha considerado ésta la forma más apropiada. Inicialmente se creaba el objeto Intent simplemente, pero la versión final incluye unos extras, los cuales sirven para pasar parámetros de una clase a otra a través del objeto Intent.

```
Intent i = new Intent(SuperPang.this, MenuPang.class);
i.putExtra("sonar",sonar);
i.putExtra("vibrar",vibrar);
i.putExtra("vidas",vidas);
startActivity(i);
```

Figura 3.25. Crear Intent.

Este procedimiento es análogo al utilizado para pasar de la clase SuperPang tanto a la clase Opciones como a Instrucciones, excepto la inclusión de extras. En este caso, la actividad SuperPang no se quiere destruir puesto que cuando la partida finaliza o el usuario la abandona, se vuelve a dicha actividad.

La clase MenuPang es la encargada de mostrar al usuario las dos opciones de juego que tiene. Cuando se selecciona una de ellas, llama a la actividad elegida creando un nuevo Intent y posteriormente se cierra la actividad MenuPang con la llamada a finish() (figura 3.26) puesto que ya ha realizado su función y no se necesita hasta un nuevo juego.

```
Intent j = new Intent(MenuPang.this, GameTour.class);
startActivity(j);
finish();
```

Figura 3.26. Finalizar actividad.

La llamada al método finish() también se ha incluido en la clase SuperPang en el caso en el que el usuario abandone definitivamente la aplicación. De igual manera sucede durante la partida, en cuyo caso será la actividad GameTour o GamePanic la que se cierre.

Estas dos clases, durante su creación, crean un objeto Intent encargado de iniciar no una actividad, sino un servicio, en este caso la clase MusicaDeFondo, la cual reproduce música en background. La figura 3.27 muestra el procedimiento en este caso.

```
Intent i = new Intent(getBaseContext(),MusicaDeFondo.class);
i.putExtra("pantalla", level);
startService(i);
```

Figura 3.27. Iniciar servicio.

Otra manera distinta de cambiar de actividades se ha implementado en las clases RankingTour y RankingPanic, que en según qué situación, cargan una tercera clase, PuntuacionDialog. La creación del Intent no reviste cambios pero la llamada a la nueva actividad si se realiza de forma diferente. En las clases que muestran el ranking de puntuaciones se utiliza el método startActivityForResult(Intent j, int id) que carga la clase PuntuacionDialog a la espera de una respuesta. Dicha clase debe implementar el método setResult(int code, Intent i) con la respuesta deseada, que es posteriormente capturada en las clases ranking en su método onActivityResult(int requestCode, int resultCode, Intent data).

3.3.3. Interfaz de usuario

En este apartado se muestra la implementación de las diferentes pantallas del juego. Algunas de ellas utilizan archivos xml que definen la interfaz y otras utilizan el canvas del SurfaceView para mostrar la interfaz.

Layouts

Los layouts conforman la arquitectura del interfaz de usuario de una Activity [21]. Definen la estructura del interfaz y contiene todos los elementos que se muestran al usuario. Los layouts se pueden crear de dos formas:

- **Declarando los elementos del interfaz de usuario en un archivo xml:** Android proporciona un sencillo vocabulario XML correspondiente a las clases y subclases de View. Esta modalidad ha sido escogida para los layouts del menú principal, opciones, instrucciones, pantalla de selección de modos y pantalla de máximas puntuaciones.

- **Instanciando los elementos del layout en tiempo de ejecución:** Se puede crear objetos View y ViewGroup mediante código. Esta opción no ha sido utilizada.

Por ejemplo para la clase inicial, SuperPang, se define un layout vertical con un ImageView a modo de titulo del juego y tres Button que muestran las tres opciones que el usuario puede escoger. Estos tres botones aparecen organizados en un layout horizontal acompañados de sus respectivos ImageView que muestran una pelota roja y sirven para mostrar la opción que tiene el foco de selección. El archivo xml que representa este interfaz se muestra en la figura 3.28.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:orientation="vertical" android:gravity="center"
    android:background="#ff060a7d" android:id="@+id/fondojuego" >
    <ImageView android:id="@+id/imagen_fondo" android:src="@drawable/superpang"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:padding="10sp" />
    <LinearLayout android:orientation="horizontal"
        android:layout_height="wrap_content" android:layout_width="fill_parent"
        android:gravity="center">
        <ImageView android:id="@+id/bola1" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/pelota_roja_pequena" />
        <Button android:id="@+id/game_start" android:text="@string/game_start"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:layout_gravity="center" android:textColor="@color/amarillo"
            android:textSize="20sp" android:background="@android:color/transparent"/>
    </LinearLayout>
    <LinearLayout android:orientation="horizontal"
        android:layout_height="wrap_content" android:layout_width="fill_parent"
        android:gravity="center">
        <ImageView android:id="@+id/bola2" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/pelota_roja_pequena" />
        <Button android:id="@+id/opciones" android:text="@string/opciones"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:layout_gravity="center" android:textColor="@color/amarillo"
            android:textSize="20sp" android:background="@android:color/transparent"/>
    </LinearLayout>
    <LinearLayout android:orientation="horizontal"
        android:layout_height="wrap_content" android:layout_width="fill_parent"
        android:gravity="center">
        <ImageView android:id="@+id/bola3" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/pelota_roja_pequena" />
        <Button android:id="@+id/instrucciones" android:text="@string/instrucciones"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:layout_gravity="center" android:textColor="@color/amarillo"
            android:textSize="20sp" android:background="@android:color/transparent"/>
    </LinearLayout>
</LinearLayout>
```

Figura 3.28. XML menú principal.

Una vez que el archivo xml ha sido diseñado, el siguiente paso es cargarlo en la actividad. Para ello, en el método onCreate() de la clase SuperPang se utiliza el método setContentView(), pasándole como parámetro el layout en formato R.layout.main, puesto que el archivo se llama main.xml.

Este procedimiento es el que se ha seguido para cargar el resto de los diferentes layouts xml de la aplicación en la actividad correspondiente. A continuación se explican los diferentes archivos xml utilizados, mostrados en la figura 3.29, a excepción del archivo main.xml mostrado anteriormente:

- **opciones.xml**: PreferenceScreen que contiene en su interior dos CheckBoxPreference, que muestra las opciones de sonido y vibración, y un ListPreference que muestra el número de vidas inicial elegido por el usuario. Utilizado en la pantalla de opciones.

- **instrucciones.xml**: ScrollView con varios TextView en su interior almacenados dentro de un LinearLayout vertical. Los diferentes TextView muestran las instrucciones del juego.

- **menu.xml**: Muestra el menú de selección de modo de juego. Se compone de un LinearLayout horizontal en cuyo interior se encuentra un LinearLayout vertical. Dentro del layout vertical se encuentran un TextView con el texto "SELECCIONA MODO", un TableLayout con dos filas y tres columnas y por ultimo un TextView animado que recorre la pantalla de derecha a izquierda.

  En el TableLayout se encuentran en la primera fila de izquierda a derecha, un ImageView que representa la imagen del modo de juego Panic, un TextView que muestra el contador de tiempo restante para finalizar la pantalla de selección de modo y otro ImageView que representa la imagen del modo de juego Tour. En la segunda fila, debajo de los ImageView se encuentran dos TextView con una breve descripción del modo de juego.

- **ranking_tour.xml**: Muestra la pantalla de máximas puntuaciones del modo Tour. Contiene un LinearLayout general orientado horizontalmente en cuyo interior se encuentra otro LinearLayout orientado horizontalmente que a su vez contiene dos TextView de diferente color que indican el titulo de la pantalla con las palabras "RANKING" y "TOUR".
  Dentro del primer LinearLayout y en paralelo al segundo, se encuentra otro LinearLayout con orientación vertical en cuyo interior se encuentra un TableLayout con cinco filas y cuatro columnas.

  La primera columna muestra la posición de la puntuación, la segunda muestra la puntuación obtenida, la tercera indica el identificador de puntuación escogido por el jugador y la cuarta representa el nivel de pantalla alcanzado. Cada fila es independiente del resto y todas aparecen coloreadas de diferentes colores para una mejor identificación.

- **ranking_panic.xml**: Está estructurado exactamente igual que ranking_tour.xml pero aplicado a la pantalla de máximas puntuaciones del modo Panic. Solo cambia las palabras del título de la pantalla por "RANKING" y "PANIC".

- **set_puntuacion.xml**: Es la pantalla que el usuario ve cuando debe introducir un identificador de puntuación por haber conseguido una de las máximas puntua-

ciones. Se compone de un RelativeLayout en cuyo interior se encuentran dos TextView que avisan al usuario que ha conseguido una de las mejores puntuaciones y le indican que debe introducir tres caracteres, un EditText usado por el usuario para introducir su identificador y un Button para confirmar el texto.

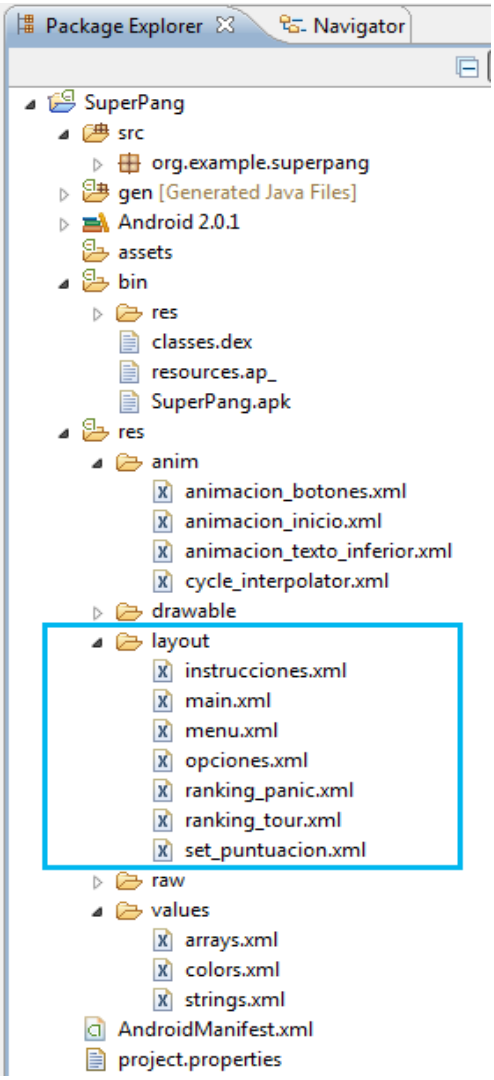


Figura 3.29. Archivos XML.

Canvas

El objeto Canvas es utilizado para mostrar el interfaz de la partida en sus diferentes modos, es decir, en GameViewTour y GameViewPanic. En las actividades GameTour y GamePanic se crea una instancia de estas clases y se establecen como vista de la actividad utilizando setContentView() de manera que lo que se muestre en GameViewTour o GameViewPanic es lo que se muestra al usuario.

Todos los recursos de imágenes se encuentran almacenados en la ruta /res/drawable y son objetos de tipo Drawable (definido como una abstracción de algo que puede ser pintado).

El primer paso es inicializar las variables correspondientes en el método inicializar(). En este método se crean dos rectángulos (figura 3.30) que limitan la pantalla en dos partes, la parte superior es donde tiene lugar la acción de la partida y su altura es 4/5 del total de la pantalla; y la parte inferior es la que contiene la puntuación, numero de vidas, nivel de pantalla, tiempo para final de partida o barra de nivel, y ocupa la quinta parte restante del tamaño de la pantalla. Se configuran los diferentes Paint para representar los textos del rectángulo inferior, se crea el objeto Personaje y según el número de pantalla o nivel, se configura el fondo de pantalla, el borde utilizado y la posición inicial de las pelotas y bloques.

```
rectSuperior = new Rect(0,0,this.getWidth(),this.getHeight()-this.getHeight()/5);
rectInferior = new Rect(0,rectSuperior.bottom,this.getWidth(),this.getHeight());
```

Figura 3.30. Rectángulos de la partida.

Una vez tenemos inicializados todos los objetos y variables iniciales, pasamos a pintarlos en pantalla. Para pintar en el canvas se utiliza el método onDraw(Canvas canvas), que es el encargado de pintar todo por pantalla.

En el juego se han usado dos tipos de imágenes, Drawable y Bitmap. Los objetos Drawable son pintados en pantalla con el método setBounds() pasándole por parámetros las coordenadas donde debe ser pintado.

En nuestro caso se ha decidido utilizar Drawable para pintar el fondo de pantalla, declarado en inicializar() (figura 3.31), sobre todo el rectángulo superior como se muestra en la figura 3.32.

```
fondo = getResources().getDrawable(R.drawable.fondo);
```

Figura 3.31. Crear fondo.

```
fondo.setBounds(rectSuperior);
fondo.draw(canvas);
```

Figura 3.32. Pintar fondo.

A cada pantalla o nivel se le ha asignado un borde con color alrededor del fondo de la pantalla. Para pintarlo, se crea un objeto Paint (objeto que contiene información de estilo y color sobre como dibujar geometrías, textos y bitmaps) con estilo Stroke (solo borde), un color y el ancho para el borde, tal como se indica en la figura 3.33.

```
borde = new Paint();
borde.setStyle(Style.STROKE);
borde.setColor(Color.BLUE);
borde.setStrokeWidth(5);
```

Figura 3.33. Crear un objeto Paint.

Para pintarlo solamente se indica al canvas, que se va a pintar un rectángulo, sus posiciones (en este caso la posición sobre rectSuperior) y se le asigna un Paint al rectángulo pintado (figura 3.34). Si el Paint utilizado posee un estilo FILL en lugar de STROKE, al pintar un rectángulo con este Paint, aparece un rectángulo con relleno en lugar de solo borde. Este caso es el utilizado para el rectángulo inferior con un relleno de color negro.

```
canvas.drawRect(rectSuperior,borde);
```

Figura 3.34. Añadir borde al fondo.

De una manera muy similar se puede pintar texto en el canvas. El método necesario es drawText() al que se le debe indicar el texto a pintar, la posición de la esquina inferior izquierda en la que colocarlo y el Paint que define el estilo del texto.

Para pintar un bitmap, primero se crea utilizando la clase BitmapFactory (que sirve para crear objetos Bitmap) desde nuestros recursos de la aplicación (figura 3.35) y posteriormente se pinta en el canvas utilizando el método drawBitmap() (figura 3.36) indicando el bitmap a pintar, la posición de la esquina superior izquierda donde se debe colocar y el Paint a utilizar. En el juego no se han utilizado Paint con los Bitmap.

```
armas = BitmapFactory.decodeResource(context.getResources(), R.drawable.indicador_arma);
```

Figura 3.35. Crear Bitmap.

```
canvas.drawBitmap(armas,
                  rectInferior.left+rectInferior.width()*4/14,
                  rectInferior.top+borde.getStrokeWidth(),
                  null);
```

Figura 3.36. Pintar Bitmap.

El resto del método onDraw() se compone de varios textos y bitmaps (vidas, indicador de arma) pintados en la zona inferior del canvas, cuyas posiciones son relativas respecto al rectángulo inferior, y varios bitmaps (personaje, pelotas, bloques) pintados en la zona superior del canvas sobre el fondo de pantalla (figura 3.37).

Figura 3.37. Canvas inicial.

En el caso de los objetos Personaje, Pelota, BloqueRompible y BloqueIrrompible, la posición elegida para pintarlos en el canvas se encuentra almacenada dentro del propio objeto.

Desde que se instancia la clase hereditaria de SurfaceView, se mantiene constancia del estado de la partida. Hay dos bitmaps que se muestran en función del estado y aparecen antes de iniciarse la partida (estado Ready) y tras conseguir una victoria (estado Win). En la figura 3.38 aparecen sendos bitmaps.

Si durante la partida se detecta algún evento como perder vida o conseguir una victoria, se modifica la variable mMode de la clase, se comprueba en el método onDraw() que se encarga de cambiar el bitmap del personaje al bitmap correspondiente y posteriormente lo pinta en pantalla.

Además de los citados bitmaps existen otros dos, pertenecientes a las clases Disparo y Objeto, que no son pintados en onDraw() si no que se pintan en su propio hilo de ejecución, sin embargo en su construcción se les pasa por parámetros el objeto canvas

de la aplicación, por lo que técnicamente se pintan sobre el mismo canvas que el resto de bitmaps.

Figura 3.38. Bitmaps ready y victoria.

Diálogos

Otro elemento importante del interfaz de usuario son los cuadros de diálogo que se muestran al usuario antes de abandonar la partida y la aplicación. Dichos cuadros aparecen cuando el usuario pulsa el botón back de su terminal bien durante la partida o bien mientras se encuentra en el menú principal como muestra la figura 3.39.

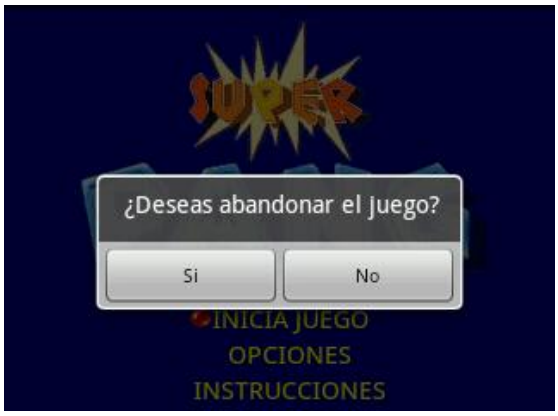


Figura 3.39. Diálogos. AlertDialog

Aunque hay varias formas de implementar un diálogo, se ha elegido la utilización de la clase AlertDialog. Esta clase extiende de la clase Dialog y puede mostrar varios botones.

Para el juego se ha implementado un diálogo que muestra un mensaje que pregunta al usuario si quiere abandonar la partida/aplicación y dos botones para cancelar o confirmar la acción.

En la implementación se ha utilizado una subclase de AlertDialog, AlertDialog.Builder, en cuyos métodos se encuentra todo lo necesario para construir un AlertDialog. Una vez implementado los elementos del diálogo, se crea el AlertDialog y se muestra al usuario como en la figura 3.40.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("¿Deseas ir al menu?")
       .setCancelable(false)
       .setPositiveButton("Si", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
               if(sonar)
                {
                    stopService(i);
                }
               Efectos.cleanup();
               finish();
           }
       })
       .setNegativeButton("No", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                gameViewTour.setState(GameViewTour.STATE_RUNNING);
                MusicaDeFondo.arrancaPlayer();
                dialog.cancel();
           }
       });
AlertDialog alert = builder.create();
alert.show();
```

Figura 3.40. Construcción AlertDialog.

Eventos de interfaz de usuario

El jugador puede interactuar con la aplicación de dos formas, bien con los botones (si su smartphone dispone de ellos) o bien mediante pulsación de la pantalla.

En las actividades que cargan una vista desde un archivo xml, están permitidos ambos tipos de interacción. El usuario se puede mover por las opciones y seleccionarlas con los botones de su dispositivo móvil y puede también seleccionarlas pulsando sobre ellas.

Esto también esta, en parte, permitido durante la partida. El usuario puede pulsar la pantalla y realizar un disparo, pero no puede realizar un disparo ni mover al personaje a través de los botones. Sin embargo, si se ofrece la posibilidad de, con los botones del terminal, abandonar a aplicación y pausar la partida.

La implementación que define que ocurre al pulsar un botón se encuentra en el método onClick(View v) del escuchador correspondiente de cada botón (figura 3.41)

```
botonStart = (Button)this.findViewById(R.id.game_start);
botonStart.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        /*
         * codigo a implementar
         */
    }
});
```

Figura 3.41. Manejar botones.

En la clase SuperPang el usuario interactúa con tres botones, diseñados con un color transparente para dar una impresión uniforme de color, mientras que en MenuPang, el usuario interactúa con dos objetos ImageButton definidos con una imagen de fondo y que tienen la propiedad de poder ser seleccionados.

Para detectar las pulsaciones de pantalla se debe implementar el método onTouchEvent(MotionEvent event) dentro de la clase correspondiente.

Este método ha sido implementado en las clases GameViewTour y GameViewPanic para detectar la pulsación del usuario en pantalla e iniciar la partida o realizar un disparo. No se restringe al usuario ninguna zona de la pantalla sobre la que pulsar. También ha sido implementado en RankingPanic y RankingTour de manera que al detectar la pulsación, se cierra la actividad y se vuelve al menú principal.

Icono

En Android es posible definir un icono que representa a la aplicación tanto en el escritorio con en el lanzador de aplicaciones. Debe ser un formato de PNG de 32 bits y tiene tres objetivos principales: promover la marca y contar la historia de la aplicación, ayudar a los usuarios a descubrir la aplicación en el Market de Android y funcionar en el lanzador de aplicaciones.

El icono debe ser distinguible a baja resolución ya que existen muchos móviles con una baja calidad de pantalla y no debe representar imágenes recortadas o demasiado finas ya que debe servir para distinguir la aplicación de las demás.

Por este motivo se ha seleccionado como icono, una de las imágenes utilizadas en la aplicación puesto que cuenta con los tres requisitos mencionados anteriormente (figura 3.42). La imagen elegida representa claramente el juego y distingue claramente la aplicación de cualquier otra.

Figura 3.42. Icono de la aplicación.

3.3.4. Núcleo del juego

En este apartado se explican detalladamente las colisiones, el movimiento del personaje, las pelotas, el disparo y los objetos con los que el personaje puede interactuar.

Colisiones

Las colisiones forman uno de los pilares básicos del juego. Se utilizan para detectar cualquier choque entre las clases Personaje, Pelota, Disparo, Objeto, BloqueRompible y BloqueIrrompible, y sin tenerlas en consideración el juego no tendría sentido.

Cada objeto de las clases anteriores tiene un parámetro de tipo Rect utilizado para detectar dichas colisiones y cuyas coordenadas se actualizan junto a la posición del objeto.

El tamaño del rectángulo es siempre levemente inferior al tamaño del objeto de manera que el bitmap engloba al rectángulo y no al revés (figura 3.43). Se ha implementado de esta forma para asegurarse que las colisiones ocurren realmente y da tiempo a mostrarlas en el canvas durante el juego. Si se establecen los rectángulos cuyos bordes coincidan con los bordes del bitmap, es posible que en dispositivos móviles sin un buen procesador, la colisión se detecte antes de pintarla sobre el canvas de manera que el usuario puede no apreciarla correctamente.

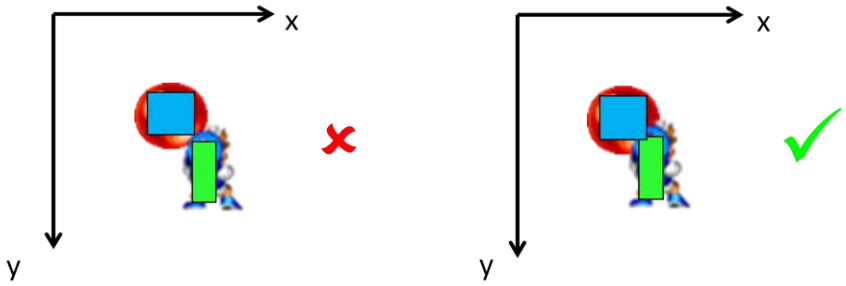


Figura 3.43. Colisión.

Para detectar la colisión entre dos rectángulos se utiliza el método intersect(Rect a, Rect b) de la clase Rect. Este método devuelve un boolean igual a true si hay intersección entre los dos rectángulos indicados y un boolean igual a false si no hay intersección.

Dichos rectángulos no se pintan ya que harían antiestética la aplicación, sin embargo en la fase de pruebas si se han necesitado pintar para comprobar que su funcionamiento es correcto.

Todas las colisiones son evaluadas dentro del método actualizarPosiciones() de las clases GameViewTour y GameViewPanic. Las diferentes colisiones son explicadas a continuación:

- **Colisión Pelota – Player:** Se produce cuando el rectángulo del objeto Pelota interseca con el rectángulo del objeto Personaje. Si el resultado de la comprobación es true, se modifica la variable mMode. Esta variable es evaluada en el método onDraw() que se encarga de llamar al método correspondiente de la clase Personaje para modificar su bitmap y llama al método eventoVida() de la propia clase perdiendo una vida.

- **Colisión Disparo – Pelota:** Se produce cuando el rectángulo del objeto Disparo colisiona con el rectángulo del objeto Pelota. Se evalúa cuando el objeto Disparo es distinto de null. Si la comprobación es true, se llama al método setColisionDisparo(true) de la clase Disparo, posteriormente se llama a explosionPelota() que se encarga de anular el disparo actual, eliminar la pelota colisionada y crear dos pelotas nuevas de tamaño inferior y con direcciones opuestas incrementando a su vez la puntuación total. Por último se emite un efecto de sonido a modo de explosión.

- **Colisión Pelota – BloqueRompible o BloqueIrrompible**: En ambos casos la situación es similar puesto que las pelotas no afectan a los bloques. Se produce cuando los rectángulos de ambos objetos colisionan. Se accede al método colisionBloquePelota() donde se determina el comportamiento de la pelota ya que dependiendo del lugar del bloque donde golpee tomara una dirección u otra.

- **Colisión Disparo – BloqueRompible:** Se evalúa cuando el disparo es distinto a null y existe algún bloque naranja en pantalla. Si la comprobación de la colisión es true se verifica si el bloque tiene asignado un icono, en cuyo caso se instancia el Objeto correspondiente, se actualiza la puntuación, desaparecen el bloque y el disparo actual y se emite un efecto de sonido.

- **Colisión Disparo – BloqueIrrompible**: Se evalúa cuando el disparo es distinto a null y existe algún bloque amarillo en pantalla. Si la colisión existe el bloque

no desaparece y dependiendo del tipo de disparo (estándar o pausa), desaparecerá o permanecerá pegado al bloque durante unos instantes. Transcurrido ese tiempo el disparo desaparece si no ha sido golpeado por una pelota.

- **Colisión Objeto – Player:** Se evalúa únicamente mientras el objeto existe, tanto si se encuentra cayendo al suelo de la pantalla como si se encuentra en él. Según el tipo de objeto se obtiene una vida extra, puntuación extra o se obtiene el disparo pausa. En cualquier caso se emite un efecto de sonido, se actualiza la puntuacion, se provoca el fin del hilo que maneja el objeto y se elimina el objeto.

Movimiento del personaje

El personaje es un objeto de la clase Personaje y visualmente se representa por un bitmap pintado en unas coordenadas X e Y almacenadas en el propio objeto. A la hora de pintar el personaje, no se pinta en la coordenada X que almacena el objeto, sino que se toma como coordenada izquierda en la que pintar el valor de la coordenada almacenada en el personaje restándole la mitad del ancho del bitmap (figura 3.44). Se realiza este procedimiento para evaluar más fácilmente la colisión del personaje con los límites de la pantalla.

```
canvas.drawBitmap(  player.getGraphic(),
                    player.getPosicionX() - player.getGraphic().getWidth()/2,
                    player.getPosicionY(),
                    null);
```

Figura 3.44. Pintar bitmap.

Solo se puede desplazar en el eje X por lo que su coordenada del eje Y no va a variar durante toda la partida. Inicialmente la coordenada en el eje X del personaje es la coordenada central en dicho eje de la pantalla, es decir el ancho total entre dos.

Antes de utilizar el sensor acelerómetro se evaluó su funcionamiento. El sensor permite acceder a la aceleración en los ejes X, Z e Y distribuidos como muestra la figura 3.45, cuyos valores vienen dados en forma de array. Los sensores del dispositivo móvil no distinguen entre orientaciones de pantalla, de manera que en nuestro caso al utilizar una orientación apaisada, debemos acceder a la coordenada Y del sensor.

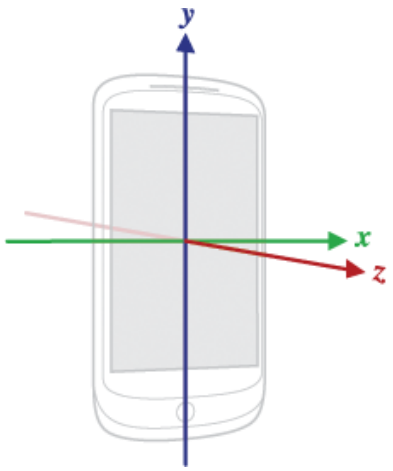


Figura 3.45. Sensor acelerómetro en Android.

Puesto que Android toma como eje de coordenadas la esquina superior izquierda de la pantalla, los desplazamientos hacia la derecha se realizan aumentando la posición X del personaje, y los desplazamientos hacia la izquierda se realizan disminuyendo dicha posición, mientras que los desplazamientos hacia abajo se realizan aumentando la posición Y, disminuyéndola en caso de desplazamiento hacia la zona superior de la pantalla.

Antes de utilizar el acelerómetro, es necesario asociar la medida del sensor a la clase en la que se va a medir como muestra la figura 3.46. Primero es necesario obtener el servicio de sensores del móvil y elegir el sensor acelerómetro de la lista total de sensores y por último se registra el sensor en la clase, indicando una tasa de lectura de datos. En nuestro caso se ha utilizado SensorManager.SENSOR_DELAY_GAME, que es la tasa mínima para utilizar el sensor en un juego. Para poder utilizarlo, la clase registrada debe implementar el interfaz SensorEventListener.

```
SensorManager sensores = (SensorManager)getSystemService(Activity.SENSOR_SERVICE);
List<Sensor> listaSensores = sensores.getSensorList(Sensor.TYPE_ACCELEROMETER);
Sensor acelerometro = listaSensores.get(0);
sensores.registerListener((SensorEventListener)gameViewTour, acelerometro, SensorManager.SENSOR_DELAY_GAME);
```

Figura 3.46. Registro del sensor acelerómetro.

Una vez iniciada la partida se mide el sensor acelerómetro dentro del método onSensorChanged() y se evalúa si el valor del acelerómetro en el eje Y es positivo o negativo produciéndose un desplazamiento hacia la derecha o izquierda respectivamente, modificando el valor de la coordenada X del personaje, en un valor constante y dependiente del ancho de pantalla, para poder adaptar el juego a diferentes resoluciones como muestra la figura 3.47. Si el sensor no muestra movimiento, el personaje permanece inmóvil.

```
if((event.values[1])<0)
{
    acelerometro = - 2*width/320;
    player.modoAnimacion = player.ANIMACION_IZQUIERDA;
}
```

Figura 3.47. Manejar acelerómetro.

La variación de dicha coordenada se realiza en el método update() del personaje al que se le pasa la anterior constante que varía su posición. En su interior se actualiza la posición del personaje, se comprueba su posición para evitar que se salga de los límites de la pantalla y se actualiza su rectángulo de colisión en el código mostrado en la figura 3.48.

```
public void update(int acelerometro)
{
    setPosicionX(getPosicionX() + acelerometro);
    if (posX - personaje.getWidth() / 2 < 0) setPosicionX(personaje.getWidth() / 2);
    if (posX + personaje.getWidth() / 2 > width) setPosicionX(width - personaje.getWidth() / 2);
    setRectanguloColision(posX - this.getGraphic().getWidth()/2+this.getGraphic().getWidth()/4,
                          posY+this.getGraphic().getHeight()/5,
                          posX + this.getGraphic().getWidth()/2-this.getGraphic().getWidth()/4,
                          posY+this.getGraphic().getHeight()-this.getGraphic().getHeight()/5);
}
```

Figura 3.48. Actualizar posición Personaje.

Este método es llamado desde el hilo principal de la partida junto al método on-Draw() de manera que constantemente, tras actualizar la posición del personaje, se pinta por pantalla.

Movimiento de las pelotas

El movimiento de los objetos de la clase Pelota es otro factor muy importante en el juego. Inicialmente se pensó en dotar a dichos objetos de un movimiento lineal pero no ofrecía una buena sensación de jugabilidad por lo que se decidió hacer un movimiento que mostrase una trayectoria parabólica.

Para representar la posición de una pelota solamente hacen falta los valores de las coordenadas X e Y, pero para actualizar su posición y dar movimiento a las mismas, se utilizan dos variables que representan las velocidades en los ejes X e Y respectivamente.

En el eje X se ha decidido dar a las pelotas una velocidad lineal. Para adaptar dicha velocidad a la pantalla de cada móvil, se utiliza un factor en función del ancho de pantalla cuyo valor es igual para todas las pelotas. La figura 3.50 muestra la llamada al método setVelocidadX() de la clase Pelota que se realiza a la hora de inicializar todas las variables. En dicha sentencia, Pelota.getVelocidadX() devuelve el valor inicial asignado a la velocidad, igual a uno, y se multiplica por un factor de escalabilidad en función del ancho de pantalla y un valor elegido igual a 320.

La elección de este valor no es aleatoria. Se debe al dispositivo móvil en que se probó el juego inicialmente. Se trata de un HTC Wildfire con pantalla QVGA 240x320. Todas las variables de desplazamiento y velocidad se han escalado respecto a esta pantalla para dotar al juego de la jugabilidad mostrada en dicho terminal sea cual sea el dispositivo en que se instale la aplicación.

Una vez asignada la velocidad en el eje X (variable vx0) para cada pelota, es necesario actualizar continuamente su posición. Para ello se utiliza el código de la figura 3.49 dentro del método update() (clase Pelota), en el cual se modifica la posición dependiendo de la velocidad y la dirección que lleva, y se actualiza en el objeto. La dirección en el eje X indica si se desplaza a la derecha, valor positivo e igual a uno, o si se desplaza hacia la izquierda, valor negativo e igual a menos uno. Utilizando esta dirección han implementado los rebotes de las pelotas con las paredes laterales de manera que al tener una velocidad constante en el eje X, con solo cambiar la dirección de desplazamiento al chocar con las paredes se consigue la sensación de rebote.

```
x = x0 + (int)vx0 * getXDirection();
x0 = x;
this._coordenadas.setX((int) x);

vy0 = vy0 + a;
y = y + vy0;
this._coordenadas.setY((int) y);
```

Figura 3.49. Actualizar posición Pelota.

En dicho código se modifica la posición X de la pelota en función de su posición y velocidad actuales y se asigna a las coordenadas del objeto (la clase Coordenadas es la que almacena la posición de la pelota). La coordenada X actual se almacena en otra variable, x0, para poder realizar el desplazamiento respecto a la posición actual.

La velocidad en el eje Y (variable vy0) se actualiza utilizando una variable 'a', con valor constante, de manera que la velocidad se comporta de manera lineal. Esta variable 'a' es igual para todas las pelotas, depende de la altura de la pantalla y es actualizada en el momento de inicializar las variables (figura 3.50). Cada pelota posee su propia velo-

cidad en el eje Y. Diferentes tamaños de pelota, tienen diferentes velocidades máximas. Esto significa que si la velocidad tiene un comportamiento lineal, a mayor velocidad máxima absoluta, mayor recorrido realiza para llegar a 0. Este es el principio utilizado para diseñar la altura de rebote de las pelotas en el suelo.

```
Pelota.setA((0.08 * height/240) );
Pelota.setVelocidadX(Pelota.getVelocidadX() * width/320);
```

Figura 3.50. Inicializar variables Pelota.

Al rebotar en el suelo, se le asigna una velocidad (máxima) negativa a la pelota, que se va reduciendo al sumarle constantemente la variable 'a' hasta llegar a 0. Cuando la velocidad llega a 0, en su siguiente iteración se vuelve positiva y continua aumentando hasta un nuevo rebote en el suelo. Realizando esta operación en cada rebote, se consigue que la pelota alcance siempre la misma altura.

La posición en el eje Y no se comporta linealmente ya que depende de la velocidad, que no se trata de un valor constante como muestra la figura 3.49, si no que se actualiza en función de 'a', fluctuando entre su valor máximo negativo, al rebotar en el suelo, y su valor máximo positivo, justo antes del rebote. Con esto se consigue que cuando la velocidad Y de la pelota sea mayor, se produzca un desplazamiento en el eje Y mayor que cuando la velocidad de la pelota se acerca al valor 0, en cuyo caso el desplazamiento es menor, dando la sensación de aceleración en el desplazamiento (figura 3.51).

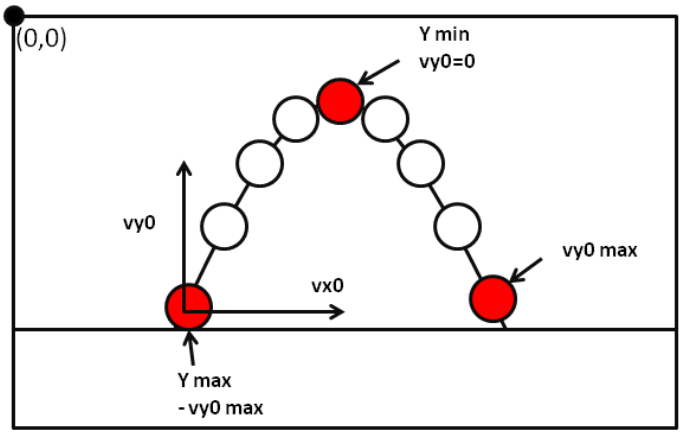


Figura 3.51. Recorrido de la pelota.

En el hilo principal de la partida, se llama al método actualizarPosiciones() de la clase que extiende de SurfaceView, en cuyo interior, se recorre el array de los objetos Pelota, y para cada una se realiza la actualización de posiciones y velocidad indicado anteriormente llamando al método update() del propio objeto.

Movimiento del disparo

Un disparo es un objeto de la clase Disparo y se representa visualmente con dos bitmaps: la punta del disparo y el tronco del disparo. Cuando se pulsa la pantalla, se toma la coordenada del eje X del personaje y se le asigna al objeto disparo. Si es la primera vez que se pulsa la pantalla, se crea el objeto y se inicia el hilo que lo controla. Dicho hilo se pone en espera (wait()) cuando el disparo finaliza y se vuelve a activar (notifyAll()) con la siguiente pulsación de pantalla.

Una vez activado el disparo, se pinta el bitmap punta encima de la base del rectángulo superior del canvas, tomando el pico del disparo como coordenada del eje Y. Se comprueba el estado de la partida, si el estado es Running, se duerme el hilo del disparo durante diez milisegundos y al despertarse se prosigue con la iteración. Si el estado es distinto a Running se duerme el hilo del disparo y cada diez milisegundos se comprueba el estado de la partida para proseguir con el disparo al reanudarla. El tercer paso es actualizar la posición del eje Y en un valor que depende de la altura de la pantalla (figura 3.52) para adaptar el movimiento a cualquier resolución. Seguidamente se actualiza el rectángulo de colisión con las nuevas coordenadas y se pinta el bitmap del tronco del disparo desde la base del bitmap punta hasta la base del rectángulo superior del canvas, dando la sensación de que el disparo sale desde abajo (figura 3.53).

```
posicionY -= 1.2*GameViewTour.staticHeight/240;
```

Figura 3.52. Actualizar posición Disparo.

```
if(posicionY <= GameViewTour.rectSuperior.bottom-disparo.getHeight()-paloDisparo.getHeight())
{
    int palos = (GameViewTour.rectSuperior.bottom - posicionY - disparo.getHeight()) / paloDisparo.getHeight();
    for(int i=0; i<palos; i++)
    {
        canvas.drawBitmap(paloDisparo, posicionX,posicionY+disparo.getHeight()+i*paloDisparo.getHeight(), null);
    }
}
```

Figura 3.53. Dibujar disparo.

Cuando el disparo llega a su fin, bien por chocar con el techo o un bloque, o bien por colisionar con una pelota, se actualiza su coordenada del eje Y a la posición inicial que tenía el disparo y se anula su rectángulo de colisión modificando sus coordenadas. Se comprueba si el usuario ha obtenido el Objeto disparo pausa (en caso afirmativo,

carga los bitmaps correspondientes para mostrarlos en el siguiente disparo) y coloca el hilo del disparo en espera como muestra la figura 3.54 en la que el hilo del disparo es la variable 't'.

```
if(tipoDisparo == SIN_DISPARO)
{

    posicionY = GameViewTour.rectSuperior.bottom - disparo.getHeight();
    setRectanguloColision(0,0,0,0);
    isShooting = false;
    setColisionDisparo(false);

    if(disparoPausa)
    {
        tipoDisparo = DISPARO_PAUSA;
        disparo = BitmapFactory.decodeResource(contexto.getResources(), R.drawable.punta_lanza2);
        paloDisparo = BitmapFactory.decodeResource(contexto.getResources(), R.drawable.palo_lanza2);
    }

    synchronized(t)
    {
        try
        {
            t.wait();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figura 3.54. Disparo en espera.

El procedimiento de actualización de posición para el disparo pausa es similar al disparo estándar. La diferencia ocurre a la hora de chocar contra el techo o un bloque irrompible. En estos dos casos se detecta la colisión y se configura el disparo para que se continúe pintando, bien apoyado en el techo o bien apoyado en el bloque, hasta que pase un intervalo de tiempo de dos segundos o una pelota golpee contra él, en cuyo caso desaparece el disparo.

Movimiento de los ítems

Nos referimos a un ítem como una instancia de la clase Objeto y se representa visualmente por un bitmap. Puede ser de tres tipos y todos tienen el mismo funcionamiento. Una vez que el jugador rompe un bloque que tiene asignado un ítem, se toman la posición central en el eje X del bloque y su coordenada Y, y son asignadas a las coordenadas del ítem en su constructor, momento en el cual se inicia el hilo propio del ítem que es quien gobierna el movimiento del mismo.

Para realizar el movimiento se comprueba que el ítem aun no ha llegado a la base del rectángulo superior del canvas y se actualiza su posición en función de la altura de la pantalla. Seguidamente se actualiza su rectángulo de colisión y se pinta en pantalla con

las posiciones actualizadas. El tercer paso es igual que el segundo del objeto Disparo. Se duerme el hilo del ítem durante diez milisegundos en cada iteración para que no alcance la base del rectángulo superior al instante, si el estado de la partida es Running; o bien se duerme el hilo y cada diez milisegundos se comprueba el estado del juego a la espera de volver al estado Running. El cuarto paso es anular el hilo del ítem en caso de entrar la partida en estado Lose (se pierde una vida) y el quinto es comprobar si el ítem ha alcanzado la base del rectángulo superior del canvas, en cuyo caso se mantiene pintando el ítem durante cinco segundos si el usuario no colisiona antes con él.

3.3.5. Otros

En este apartado se explican detalles adicionales del juego que no están explicados en los apartados principales.

Sonidos

Para el manejo de sonidos en la aplicación, además de las clases creadas MusicaDeFondo y Efecto, se utiliza la clase MediaPlayer de Android, que permite el manejo de fichero de audio y video y streams.

Los sonidos implementados ocurren en determinadas situaciones del juego:

- Al iniciarse la aplicación, junto al menú principal se emite un sonido inicial con el nombre del juego.
- Durante la pantalla de selección de modo, se reproduce una melodía que acompaña a la pantalla.
- Al perder una vida durante la partida, si la partida continua y el jugador tiene más vidas restantes se emite un carcajada por haber perdido la vida y si el jugador se ha quedado sin vidas se emite una melodía.
- Al conseguir una victoria en una pantalla o conseguir terminar un modo de juego se emite una alegre melodía.

La figura 3.55 muestra cómo crear un objeto MediaPlayer.

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.intro);
mp.start();
```

Figura 3.55. Crear MediaPlayer.

Vibración

Otro de los detalles incluidos en el menú principal y la pantalla de selección de modo es una leve vibración que el dispositivo móvil emite cuando el usuario selecciona una opción.

Para conseguir la vibración es necesario utilizar un objeto de la clase Vibrator para obtener el servicio vibrador del sistema y posteriormente llamar al método víbrate() indicando el tiempo en milisegundos que debe vibrar el dispositivo. La figura 3.56 muestra como dotar de vibración a una aplicación.

```
Vibrator vibracion = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
vibracion.vibrate(100);
```

Figura 3.56. Vibración.

Otros tipos de recursos

En el juego se han utilizado recursos de tipo Color, String y TypedArray. Estos recursos se han de colocar en archivos independientes en la ruta res/values/ y cada uno tiene una utilidad diferente.

El recurso Color se almacena en colors.xml y sirve para definir colores asignándoles el atributo 'name' como indica la figura 3.57 a través de la cual se accede al color posteriormente. De esta manera modificando el archivo xml se modifican todos los recursos que acceden a él y no es necesario modificarlos uno a uno.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="text">#ffffff</color>
    <color name="one_player_color">#aa4454</color>
    <color name="amarillo">#FFFF00</color>
    <color name="principal">#000000</color>
    <color name="azul_ranking">#2687E2</color>
</resources>
```

Figura 3.57. Colors.

El recurso String se almacena en strings.xml y sirve para definir cadenas de texto a las que se puede acceder de manera eficiente del mismo modo a como se accede a los recursos color.

El recurso TypedArray se almacena en arrays.xml y sirve para definir arrays que pueden contener cualquier tipo de objeto. En el juego se han implementado dos arrays (figura 3.58), utilizados en la pantalla de opciones por el elemento ListPreference que muestra el número de vidas de la partida. A los arrays se accede a través de su atributo 'name'. El primero de los arrays muestra las opciones que tiene el usuario mientras que el segundo representa los valores internos de cada una de las opciones siendo el valor que se usa en el juego.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="vidas">
        <item>1</item>
        <item>3</item>
        <item>5</item>
    </string-array>

    <string-array name="numero_vidas">
        <item>1</item>
        <item>3</item>
        <item>5</item>
    </string-array>
</resources>
```

Figura 3.58. Arrays.

Estos tres tipos de recursos han sido utilizados desde otros archivos xml en los atributos de las diferentes etiquetas. Para su acceso se indica el fichero que los contiene y el valor de su atributo 'name' como muestra la figura 3.59.

```
android:label="@string/mode"
android:background="@color/azul_ranking"
android:entries="@array/vidas"
```

Figura 3.59. Acceso a los recursos.

4

Conclusiones

Tras finalizar el proyecto se ha evaluado para comprobar si los objetivos iniciales han sido cumplidos. Se puede afirmar que dichos objetivos han sido logrados en su totalidad.

4.1. Implementación de un juego

El primer objetivo, realización de un videojuego para la plataforma Android, ha sido conseguido satisfactoriamente. El juego desarrollado funciona en terminales con sistema operativo Android. Además de cumplir este objetivo general se ha conseguido realizar la adaptación del videojuego "Super Pang" a dispositivos móviles.

4.2. Aprovechar las posibilidades de Android

El segundo de los objetivos planteados fue aprovechar las ventajas que ofrece Android También ha sido cumplido este objetivo ya que se ha conseguido distintas funcionalidades:

- **Pantalla táctil**: Se ha aprendido a utilizar la pantalla táctil y a interactuar con los diferentes objetos de la pantalla, además de detectar las pulsaciones y realizar alguna acción una vez pulsada la pantalla.

- **Acelerómetro**: Se ha conseguido utilizar el sensor acelerómetro y a realizar alguna acción en función del valor indicado en el sensor.

- **Botones**: Se ha aprendido a mostrar botones en pantalla, detectar su pulsación y ejecutar alguna acción tras detectar su pulsación. También se ha podido modificar su aspecto visual.

- **Cuadros de diálogo**: Se ha conseguido diseñar diferentes cuadros de diálogo para permitir al usuario abandonar la partida o aplicación.

- **Services**: Se ha implementado un servicio para la música de fondo de la partida. Se ha aprendido a lanzar el servicio e interactuar con él eligiendo cuando debe sonar la música.

- **Activities**: Se ha comprendido que para mostrar una pantalla es necesario un Activity, aprendiendo a cómo gestionar las diferentes actividades entre ellas y por separado y se han utilizado diferentes formas de cargar una vista en las actividades.

- **Imágenes**: Se han utilizado diferentes formas de mostrar imágenes, tanto por xml como por código, y de diferentes tipos (Drawable o Bitmap).

- **Sonidos**: Se han implementado diferentes tipos de sonido para diferentes situaciones. Las melodías durante la partida se han configurado en bucle repitiéndose cuando finalizan, mientras que los diferentes efectos de sonido y melodías ejecutadas fuera de la partida no se ejecutan en bucle.

- **XML**: Android soporta archivos de configuración xml, lo cual facilita la creación de interfaces gráficos y acceso a diferentes recursos. Debido a las ventajas que ello supone se ha decidido profundizar en el lenguaje y aprovecharlas.

4.3. Futuros desarrollos y ampliaciones

Se considera que este proyecto puede servir como base y ayudar a aquellos estudiantes que deseen iniciarse en el desarrollo de aplicaciones o juegos en Android.

También se ha querido dotar al juego de una funcionalidad básica que lo hicieran completamente jugable. Este objetivo se ha cumplido y se permite que futuros desarrolladores mejoren o amplíen su funcionalidad. Estos aspectos se tratan en el apartado 5, Líneas futuras.

4.4. Otras conclusiones

Aparte de los objetivos ya citados, se ha profundizado en el uso del lenguaje UML y las posibilidades que ofrece en el diseño de la aplicación. Con este lenguaje es posible rea-

lizar un diseño minucioso de las diferentes clases, casos de uso, etc. lo cual proporciona una gran ayuda en la posterior fase de implementación del juego.

A nivel personal ha resultado muy gratificante poder desarrollar un juego de principio a fin en una plataforma con la que no se estaba familiarizado. Una vez habituado a dicha plataforma y el entorno de desarrollo se espera poder seguir desarrollando aplicaciones y juegos, y su posterior comercialización, ya que a pesar de las dificultades encontradas en el camino, merece la pena el resultado final conseguido.

5

Líneas futuras

Debido a una limitación existente en el tiempo para la realización del proyecto, no se han podido implementar todas las funcionalidades deseadas que hicieran de este juego una aplicación más completa. Por este motivo, en este capítulo, se abre la posibilidad de incluir dichas funcionalidades en un futuro con el objetivo de poder hacer disfrutar más al usuario y alargar la vida de la aplicación.

5.1. Multijugador

A pesar de que la versión original del juego Super Pang fue diseñada para dos jugadores, la implementación de este proyecto está basada en la jugabilidad de un solo jugador.

Debido a que la opción de dos jugadores en un solo terminal es inviable, se podría implementar dicha funcionalidad permitiendo a cada usuario utilizar su propio smartphone.

Una opción para permitir la comunicación entre los distintos terminales es el uso del Bluetooth. Android permite utilizar el API Bluetooth para poder comunicar terminales distintos sin el uso de cables. Un procedimiento para el uso del Bluetooth podría ser activar el Bluetooth, buscar el dispositivo dentro del radio de alcance, establecer comunicación y transferir los datos necesarios para jugar. Esta opción tiene la ventaja de no necesitar de una conexión de datos y ser gratuita.

Otra posible opción es realizar la conexión a través del protocolo IP, en cuyo caso ambos dispositivos deberán conectarse entre sí conectándose a la red de datos. La ventaja de esta opción es que ofrece más seguridad de transferencia de datos que Bluetooth pero es inviable sin una conexión de datos de pago o sin una red WiFi.

5.2. Incremento de niveles o pantallas

El modo Tour del juego se compone de 4 pantallas y el modo Panic permite al usuario alcanzar el nivel 15. Una opción interesante para mejorar la funcionalidad de la aplicación podría ser la creación de un generador de niveles automático para alargar la vida del juego. Un punto a favor del generador de niveles es que al generarse aleatoriamente cada partida sería diferente al resto de partidas jugadas.

Si se declina esta opción también se pueden diseñar por código todos los niveles lo que podría hacer el juego algo repetitivo aunque igualmente alargaría su duración.

5.3. Selección de niveles

Otra mejora podría ser implementar una actividad que mostrase una lista desplegable con los números de pantalla para que el usuario eligiera el número que desee. Podría ser una buena opción en caso de diseñar los niveles por código ya que los usuarios que se queden atascados en un nivel o pantalla pueden conocer el resto.

Otra opción a incluir relacionada con la selección de niveles podría ser diseñar un editor que permita al usuario diseñar sus propios niveles.

5.4. Otras mejoras

Otros pequeños detalles que podrían hacer más completo el juego son:

- Incluir más variedad de ítems como protectores de vida, armas nuevas, etc.
- Incluir variedad en las pelotas de la pantalla como nuevos colores o nuevas formas de pelota con un movimiento diferente.
- Incluir algún elemento externo que modifique el desarrollo de la partida como otro personaje que ayude durante unos segundos.
- Añadir un segundo personaje controlado por el sistema otorgándole cierta inteligencia artificial.

6

Fase de Testing

En el desarrollo de cualquier aplicación es muy importante realizar una fase de pruebas que sirva para evaluar el funcionamiento de la aplicación y detectar sus puntos fuertes y débiles. En esta fase se analiza la opinión que tienen los usuarios respecto al juego y el impacto que la aplicación provoca en ellos.

Además de la opinión personal de cada encuesta, se analiza un concepto clave para cualquier videojuego, la jugabilidad.

6.1. La jugabilidad

La jugabilidad es un factor importante en la realización de cualquier videojuego y sirve para describir la calidad del juego tanto por su funcionamiento como por su diseño. Este concepto no existe como tal dentro del marco de términos admitidos por la RAE, aunque un grupo de investigadores de la Universidad de Granada ha propuesto una (muy acertada) definición del término: "conjunto de propiedades que describen la experiencia del jugador ante un sistema de juego determinado, cuyo principal objetivo es divertir y entretener de forma satisfactoria y creíble, ya sea solo o en compañía." [22].

6.2. Propiedades de la jugabilidad

Para medir la jugabilidad es necesario definir varios factores que proporcionan información muy válida para evaluar el proyecto. Estos factores tienen peso por si solos pero es necesario evaluarlos en conjunto para obtener un resultado concluyente en el análisis.

Según la tesis doctoral de José Luis González Sánchez, miembro de la Universidad de Granada, se pueden distinguir varios factores para definir más detalladamente la experiencia del jugador [23].

6.2.1 Satisfacción

Se define como nivel de agrado del jugador una vez que ha completado el juego, es decir, si le gusta o no. Es la propiedad más importante ya que en última instancia, el objetivo de un videojuego es divertir y entretener al usuario final. Si se consigue es muy posible que el videojuego tenga éxito.

Es un factor de difícil medición ya que tiene una alta carga subjetiva y depende de a que publico está orientado el videojuego. No es igual la satisfacción que puedan sentir un niño y un adulto con diferentes tipos de juego adecuados a su edad y madurez, como por ejemplo un juego infantil y un juego extremadamente violento.

Aunque la satisfacción es un atributo global, existen varios parámetros que componen dicha valoración y se pueden definir por separado:

- Diversión: Es el objetivo fundamental de cualquier videojuego ya que si no es capaz de divertir al usuario, no será capaz de satisfacerlo.

- Placer: Además de divertir y entretener, un juego debe ser capaz de dar placer al usuario. No se puede permitir la frustración del usuario ya que eso degenera en malestar al jugar y un posterior abandono del juego.

- Atractividad: A nivel general el juego debe resultar atractivo al usuario, no solo estéticamente sino también a nivel argumental, narrativo, musical, etc.

6.2.2. Aprendizaje

Representa la facilidad para comprender y dominar el sistema del juego y su mecánica, es decir, el tiempo que debe emplear el usuario en adaptarse y dominar el sistema del juego y entender sus objetivos.

Para ello deben mostrarse al usuario un correcto sistema de menús, controles e interacción para permitir al usuario saber en todo momento a que juega y como debe jugar. Inicialmente el juego debe ser asequible para cualquier jugador pero se debe adquirir mucha práctica para completar el juego totalmente.

Los parámetros que afectan a esta propiedad son:

- Conocimiento del juego: Influido por el grado de conocimiento en el juego, su mecánica y sus reglas, de manera que un jugador habituado a un genero de juego o a juegos similares, tendrá más experiencia en este campo y requerirá un menor tiempo de aprendizaje.

- Habilidad: Una vez comprendida la mecánica y las reglas, el jugador interactúa con el juego. De su habilidad para completar los diferentes retos depende el grado de aprendizaje necesario.

- Dificultad: El grado de dificultad influye en el grado de aprendizaje y por lo general a mayor dificultad más tiempo se debe invertir en dominar el juego. Es importante adaptar correctamente la dificultad para motivar al jugador para evitar su frustración.

- Frustración: Sensación que aparece en el usuario al sentirse incapaz de superar el reto propuesto. Es necesario evitar la aparición de este sentimiento ya que, en general, provoca la perdida de interés y abandono del juego.

- Velocidad: Es necesario introducir la información y mecánica del juego a un ritmo adecuado a la dinámica del juego para evitar que el usuario se aburra o sature. Al acertar con la velocidad, el usuario no es consciente de su aprendizaje y se sumerge en el hilo argumental sin darse cuenta.

- Descubrimiento: Es necesario mostrar el sistema de juego al usuario mediante guías o tutoriales, de manera que asimile mejor la mecánica del juego y necesite menos tiempo de aprendizaje.

6.2.3. Efectividad

Tiempo y recursos que son necesarios para ofrecer diversión al jugador mientras avanza en el videojuego y alcanza el objetivo final.

Un videojuego efectivo es aquel que capta la atención y el interés del jugador desde el primer momento y consigue mantenerlo entretenido hasta su finalización. Es necesario conseguir esto, aun incluso en juegos de larga duración. Para ello es recomendable ofrecer retos y objetivos secundarios que incentiven y diviertan al usuario.

Los parámetros asociados a esta propiedad son:

- Completitud: Si un jugador ha sido capaz de completar el juego al 100% (incluidos objetivos secundarios) se entiende que éste ha sido efectivo por mantener el interés del jugador hasta el final.

- Estructuración: Un juego está bien estructurado cuando sus elementos también lo estan, es decir, hay un equilibrio entre los objetivos a conseguir y las herramientas que se proporciona al jugador para conseguirlos. Una mayor estructuración conlleva un aumento del grado de efectividad del juego.

6.2.4. Inmersión

Es la capacidad de conseguir que el jugador se integre en el mundo virtual mostrado en el juego volviéndose parte de éste e interactúe con él. Es muy importante que el jugador se sienta parte del mundo virtual al que está jugando.

Los parámetros asociados a esta propiedad son:

- Conciencia: Como de creíble es percibido el mundo virtual por el jugador, tanto visual como sentimentalmente, lo que provocará una mayor implicación por parte del jugador.

- Concentración: Es importante conseguir que el jugador se mantenga concentrado durante todo el juego. A mayor concentración, mayor implicación.

- Realismo: Es más fácil conseguir que el jugador se sumerja en el mundo virtual cuanto más real sea dicho mundo.

- Destreza: Se trata de la habilidad del jugador para realizar movimientos y acciones en el mundo virtual. No es controlable, pero influye bastante en la inmersión del jugador en el mundo virtual.

- Cercanía sociocultural: Es importante desarrollar un videojuego conociendo el grupo social al que va destinado. Acertar en este contexto ayuda a una mayor inmersión por parte de los jugadores.

6.2.5. Motivación

Es la característica del juego que mueve al jugador a realizar diversas acciones y persistir en ellas para lograr ciertos objetivos.

Los parámetros que influyen en esta propiedad son:

- Estimulación: Es importante conseguir disminuir los niveles de frustración en el jugador para que tenga confianza en poder alcanzar los objetivos y que siga jugando.

- Curiosidad: Hay que conseguir despertar en el jugador la sensación de curiosidad o intriga por saber lo que ocurrirá después.

- Automejora: Tiene que ver con la mejora del personaje controlado por el jugador durante el juego o la adquisición de nuevas habilidades.

- Diversidad: La existencia de una amplia variedad de objetivos, habilidades, herramientas, etc. hace menos monótono el juego y motiva al usuario a conseguir alcanzarlos.

6.2.6. Emoción

La emoción es un impulso involuntario en el jugador debido a los diferentes estímulos del videojuego. Conseguir despertar en el jugador diversas emociones o sentimientos es muy importante ya que consigue atraer mucho más al jugador.

Los parámetros que influyen en esta propiedad son:

- Reacción: La reacción de los jugadores desencadena distintos tipos de emociones relacionadas con su desarrollo emocional personal.

- Conducta: Los videojuegos que consiguen modificar la conducta del jugador a lo largo del juego, son aquellos que consiguen transmitir mas emociones.

- Atracción sensorial: Para la transmisión de emociones del videojuego al usuario son necesarios unos canales de estimulación, a través de los cuales el desarrollador logra provocar en el jugador las emociones que desea.

6.3. Fase de pruebas

Para realizar un análisis más profundo del desarrollo del juego se ha realizado una breve encuesta entre un grupo de personas voluntarias acerca de la jugabilidad del juego y su

opinión sobre el mismo. Con los resultados obtenidos se puede analizar la impresión general que proporciona el juego así como detectar sus puntos débiles y fuertes.

6.3.1. Test de jugabilidad

En este estudio se ha agrupado a todos los encuestados en un mismo grupo ya que se trata de un juego apto para todas las personas independientemente de sus características.

La encuesta se divide en dos partes:

- **Análisis de la jugabilidad**: En este apartado se analizan las características explicadas en el apartado 6.2. Cada pregunta se valora desde un "1" si la opinión del encuestado es muy desfavorable respecto al aspecto en cuestión, hasta un "5" si su opinión es muy favorable.

- **Análisis personal**: En este apartado se le pide al encuestado su opinión sobre los aspectos que más y menos le han gustado, que cambios realizaría y una valoración numérica general del juego.

A continuación se muestran los resultados obtenidos en el análisis de la jugabilidad mediante una serie de tablas y gráficas en las que se representan las puntuaciones recibidas por cada propiedad y seguidamente se muestra el análisis de los puntos fuertes y débiles y las conclusiones obtenidas tras el estudio.

Característica: Satisfacción		
Nombre	**Pregunta 1**	**Valoración total**
Cesar	5	5
Antonia	5	5
Marta	4	4
Irene	4	4
Pablo	4	4
Alberto	3	3
Daniel	5	5
Jonathan	4	4

Tabla 6.1. Propiedad satisfacción. Resultados.

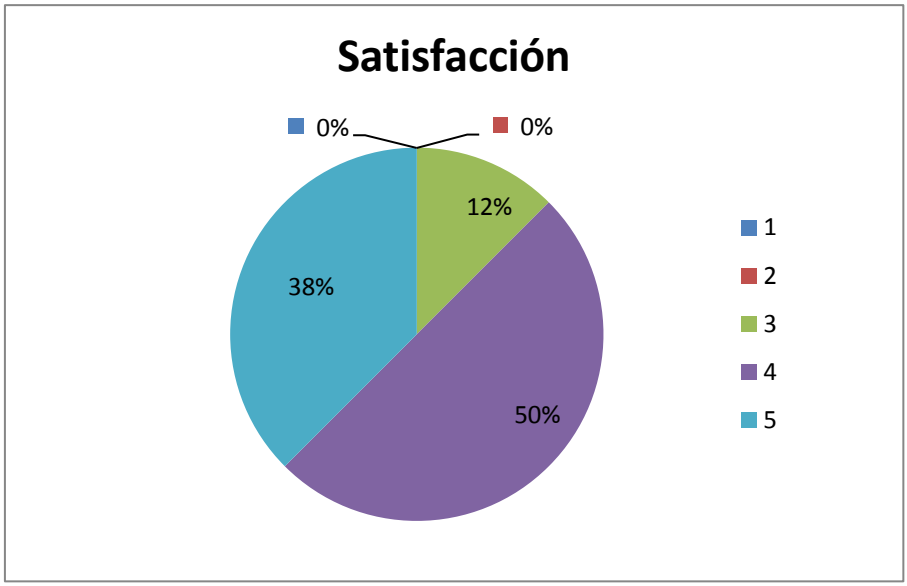


Figura 6.1. Propiedad satisfacción. Resultados.

Como se puede apreciar, tanto en la tabla de resultados como en la gráfica anterior, ninguno de los encuestados ha valorado su satisfacción por debajo de '3', un 50% ha tenido un grado de satisfacción alto y un 38% ha tenido un grado de satisfacción muy alto, por lo tanto la satisfacción se puede considerar uno de los puntos fuertes del juego.

Característica: Aprendizaje			
Nombre	**Pregunta 1**	**Pregunta 2**	**Valoración total**
Cesar	3	4	4
Antonia	3	3	3
Marta	5	3	4
Irene	4	1	3
Pablo	4	1	3
Alberto	4	3	4
Daniel	4	3	4
Jonathan	3	3	3

Tabla 6.2.Propiedad aprendizaje. Resultados

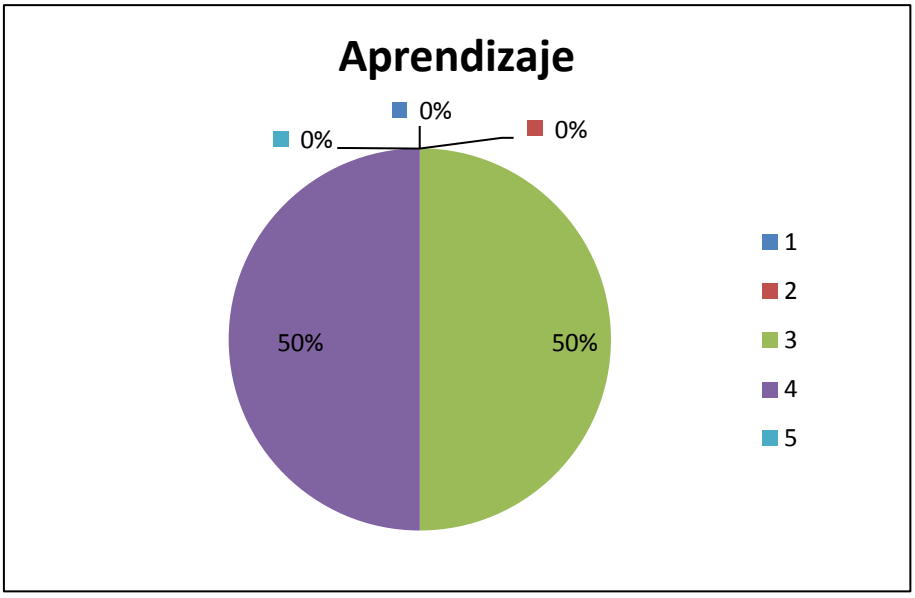


Figura 6.2. Propiedad aprendizaje. Resultados

Analizando las puntuaciones obtenidas nos damos cuenta que de las dos preguntas realizadas, la pregunta 1 recibe una valoración notablemente superior. De ello se deduce que a los usuarios les resulta fácil adaptarse a la mecánica del juego pero opinan que el juego en algunos casos es bastante difícil. Este último aspecto deberá mejorarse ya que puede provocar la frustración del jugador y su posterior abandono del juego.

Observando los resultados generales, la propiedad aprendizaje obtiene una calificación bastante aceptable, ya que la mitad de los usuarios la califican con un '3' y la otra mitad con un '4'. Si bien no resulta un punto excesivamente fuerte, tampoco se considera un punto débil por lo que esta propiedad cumple su función.

Característica: Efectividad		
Nombre	**Pregunta 1**	**Valoración total**
Cesar	3	3
Antonia	3	3
Marta	3	3
Irene	5	5
Pablo	3	3
Alberto	3	3
Daniel	3	3
Jonathan	3	3

Tabla 6.3. Propiedad Efectividad. Resultados.

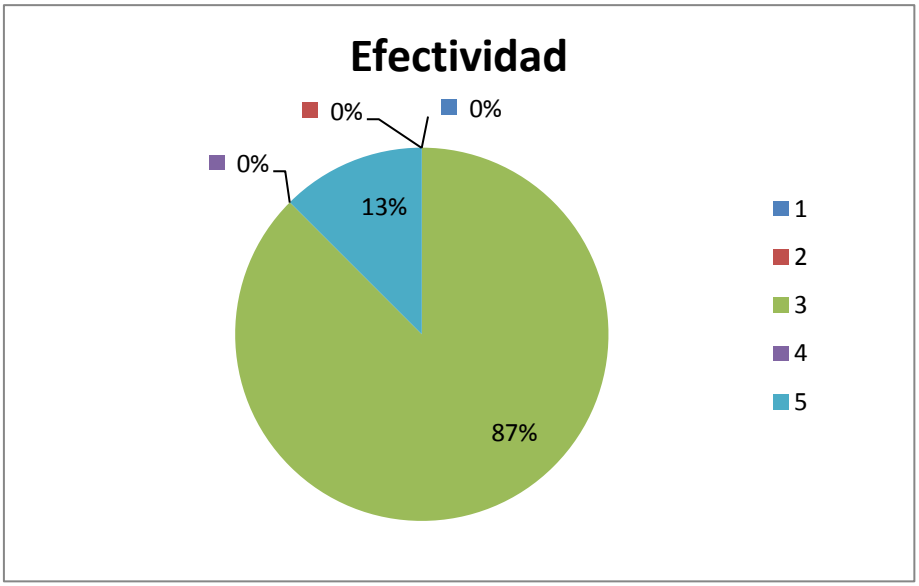


Figura 6.3. Propiedad efectividad. Resultados.

Esta propiedad ha recibido una puntuacion bastante mediocre por parte de los jugadores ya que un 87% ha optado por valorarlo con un '3' y solo a una persona le ha resultado muy satisfactoria.

A pesar de no resultar un juego corto, se considera uno de los puntos débiles del juego y se debería tener en cuenta aumentar la duración del mismo para poder satisfacer a los jugadores.

Característica: Inmersión				
Nombre	**Pregunta 1**	**Pregunta 2**	**Pregunta 3**	**Valoración total**
Cesar	4	5	5	5
Antonia	4	5	5	5
Marta	5	5	5	5
Irene	5	5	5	5
Pablo	4	4	5	4
Alberto	5	4	4	4
Daniel	5	5	5	5
Jonathan	5	4	4	4

Tabla 6.4. Propiedad inmersión. Resultados.

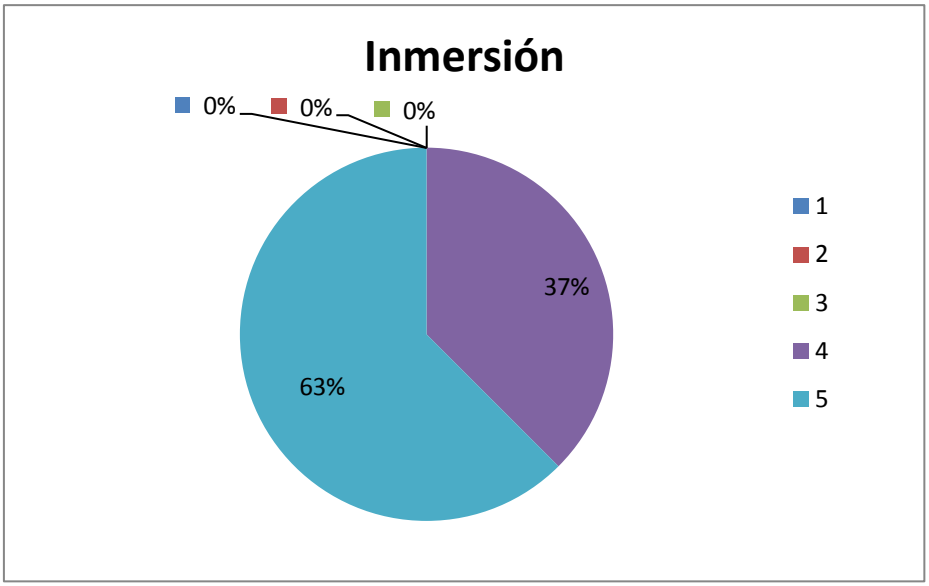


Figura 6.4. Propiedad inmersión. Resultados

Como se aprecia en los resultados, el juego es muy atractivo visualmente ya que el 63% ha puntuado esta propiedad con un '5' y el 37% restante la ha puntuado con un '4', lo que significa que resulta muy agradable a la vista y es capaz de captar la atención del jugador.

Con los buenos resultados obtenidos, esta propiedad se considera uno de los puntos fuertes del juego.

Característica: Motivación		
Nombre	**P1**	**Valoración total**
Cesar	5	5
Antonia	5	5
Marta	5	5
Irene	3	3
Pablo	4	4
Alberto	5	5
Daniel	5	5
Jonathan	2	2

Tabla 6.5. Propiedad motivación. Resultados.

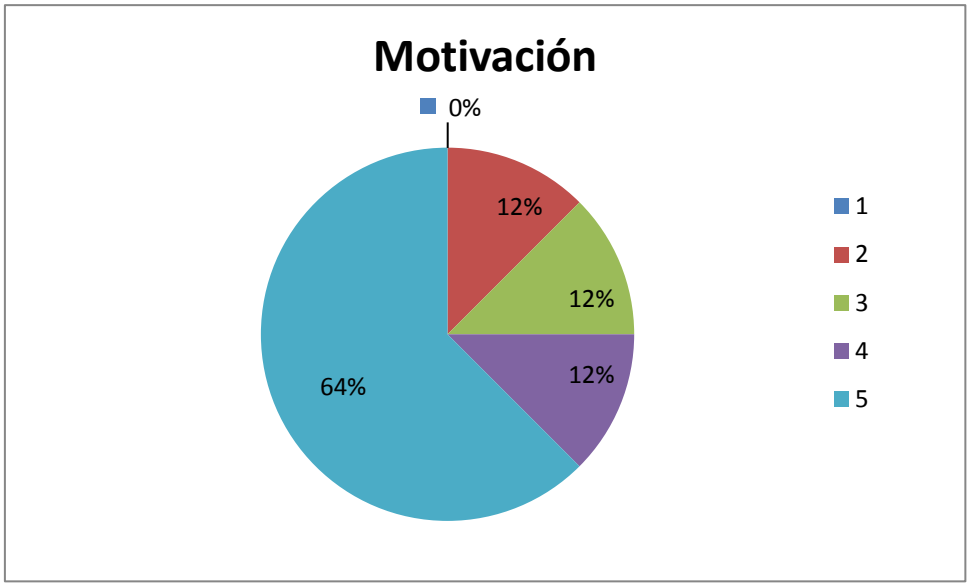


Figura 6.5. Propiedad motivación. Resultados

Según la mayoría de los usuarios, el juego es capaz de enganchar y motivar a los jugadores a terminarlo ya que un 64% de ellos ha valorado esta propiedad con un '5'. Del porcentaje restante, un jugador la ha valorado con un '2'. Esta opinión desfavorable puede ser atribuible a la propia personalidad del jugador a quien puede no motivarle este tipo de juegos o bien a la aplicación ya que al tratarse de un juego arcade, puede no conseguir enganchar a los jugadores que busquen un alto grado de realismo. De cualquier forma el resultado de esta propiedad es bastante satisfactorio.

Característica: Emoción				
Nombre	**Pregunta 1**	**Pregunta 2**	**Pregunta 3**	**Valoración total**
Cesar	4	4	3	4
Antonia	4	4	4	4
Marta	4	4	5	4
Irene	5	4	2	4
Pablo	5	5	5	5
Alberto	5	4	4	4
Daniel	4	3	3	3
Jonathan	3	3	3	3

Tabla 6.6. Propiedad emoción. Resultados.

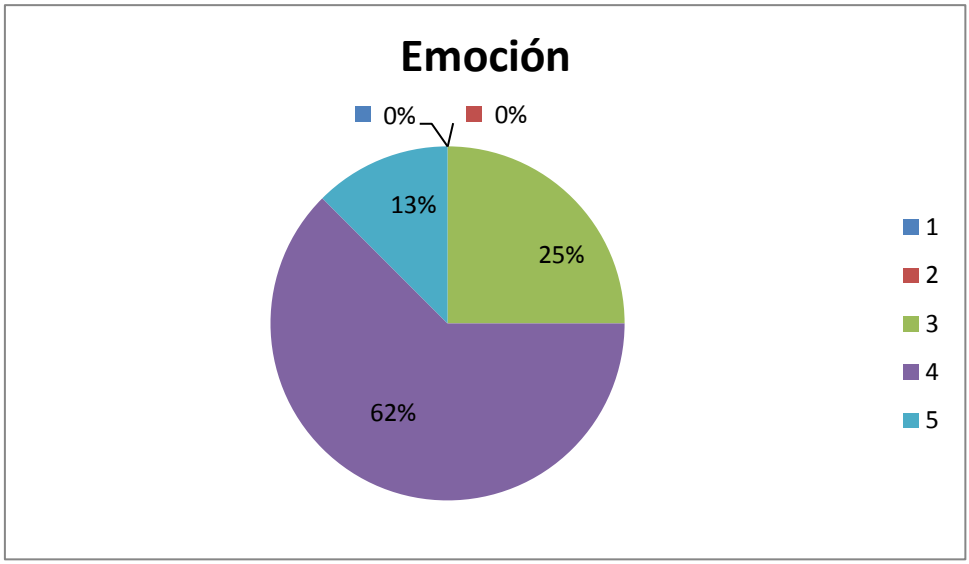


Figura 6.6. Propiedad emoción. Resultados.

Observando los resultados hay variedad de opiniones, sin embargo el 87% de los encuestados ha valorado esta propiedad con las dos puntuaciones más altas. Esta propiedad no es controlable por el desarrollador ya que depende en gran parte de la personalidad del jugador.

En la tabla 6.7 mostrada a continuación se pueden observar los puntos fuertes y débiles del videojuego.

Propiedad	Valoración. Punto débil/fuerte	Mejoras
Satisfacción	FUERTE	-
Aprendizaje	-	A pesar de no tratarse de un punto débil se puede optar por reducir la dificultad del juego para que el usuario no desista rápidamente.
Efectividad	DÉBIL	Aumentar la duración del juego en sus dos modalidades.
Inmersión	FUERTE	-
Motivación	-	Mismo caso que el aprendizaje. Se pueden incluir más variedad de items y disparos. También se pueden incluir trucos desbloqueables al terminar el juego.
Emoción	-	La mejora de las propiedades anteriores puede influir positivamente en esta propiedad.

Tabla 6.7. Evaluación general de las propiedades. Puntos débiles y fuertes.

De este análisis se pueden sacar varias conclusiones:

- Las propiedades satisfacción e inmersión son las que mayor puntuacion han recibido por lo que se puede afirmar que el juego es capaz de captar la atención del jugador y de divertirle. Es un punto muy a favor del videojuego ya que una de las principales finalidades de un videojuego es la de entretener y divertir al jugador.

- La propiedad efectividad es la que menor puntuacion ha recibido. De ello se deduce que es necesario aumentar la duración del juego puesto que si a los jugadores les parece corto, es posible que al acabarlo rápidamente pierdan el interés en él y dejen de jugarlo. También es recomendable incluir trucos o extras que se añadan al juego en sucesivas partidas una vez se ha completado de manera que el jugador siga teniendo interés por jugarlo, bien por probar dichos trucos o bien por descubrir trucos nuevos.

- El resto de propiedades han recibido una puntuacion intermedia y no se consideran ni puntos débiles ni puntos fuertes, sin embargo son susceptibles de ser modificadas para conseguir una mejora global de la aplicación.

A continuación se muestran los resultados obtenidos en el análisis personal indicando las características que más y menos han gustado a los encuestados. También se muestran los aspectos que opinan deberían mejorarse o cambiarse y la valoración general que han dado al juego.

- **Aspectos positivos**: Algunos de los aspectos que más han gustado entre los encuestados han sido la diversión y adicción que provoca a pesar de ser un juego muy sencillo. Esta opinión es muy favorable ya que a sabiendas de que la mecánica del juego es muy sencilla, se ha conseguido divertir y enganchar al jugador.

  También han destacado el gran parecido del juego al juego original tanto de maquinas recreativas como de videoconsolas, y es que se ha procurado diseñar el juego de manera que sea fácilmente reconocible como una adaptación móvil del juego original, conservando la esencia del mismo.

  Otro aspecto positivo ha sido conseguir una buena adaptación a los terminales móviles. Se ha podido adaptar correctamente a dispositivos de reducido tamaño a pesar de sus limitadas pantallas y procesadores, lo cual es un punto muy favorable hacia el videojuego.

  Por último, también han destacado la originalidad de utilizar el acelerómetro en el videojuego, y es que este punto era uno de los objetivos iniciales del juego ya qué se buscaba aprovechar las diferentes ventajas que ofrece Android, y el uso de sensores, en este caso el acelerómetro, es una de ellas.

- **Aspectos negativos**: De entre los aspectos que menos han gustado a los encuestados, destacan la dificultad de controlar al personaje con el acelerómetro, el número de pelotas en pantalla y su velocidad.

  Si bien la utilización del acelerómetro es una idea original, también puede resultar difícil de manejar debido a que es algo novedoso en el mundo de los videojuegos y a los usuarios les puede resultar algo complicado adaptarse inicialmente.

  Por otro lado la cantidad de pelotas y la velocidad a la que se mueven han parecido excesivas en algunos casos y es posible que estos factores hayan influido en otro de los aspectos calificados negativamente, la dificultad del juego, la cual ha resultado algo elevada para los encuestados.

  Otro aspecto que no ha gustado a todos los jugadores ha sido la variedad de disparos ya que los dos tipos de disparos diferentes existentes en el juego han parecido insuficientes a algún jugador.

  Por último a alguno de los encuestados le hubiera gustado poder jugar con el teclado además de tener la posibilidad de utilizar el acelerómetro.

- **Aspectos mejorables**: Los aspectos que han señalado los jugadores como mejorables en su mayoría han resultado ser detalles lo cual muestra que salvo dichos detalles, en general, estan satisfechos con el resultado final de la aplicación.

  Algún usuario ha pedido aumentar la dificultad del primer nivel, ya que le ha resultado demasiado fácil, sin embargo otros usuarios se han quejado de la dificultad general del juego.

  Otro aspecto mejorable son los disparos del juego. Algunos encuestados han pedido poder disparar más veces seguidas para poder romper mas pelotas y otros piden elegir entre un disparo simple y doble.

  También les parecería bien a los encuestados la posibilidad de añadir más items como, por ejemplo, un protector para la vida de manera que estando en poder de dicho ítem, al golpear una pelota en nuestro personaje, no signifique la perdida instantánea de una vida.

  Otro detalle en el que se ha fijado algún jugador son los fondos de las pantallas. Si bien se han elegido fondos similares al juego original algún jugador preferiría ver fondos de paisajes reales.

Por último se ha solicitado la posibilidad de mover el personaje por la pantalla manejándolo con el dedo en lugar de con el acelerómetro.

- **Valoración general**: Los encuestados han valorado el juego con una puntuacion numérica entre ‘0’ y ‘10’ tras haber probado el juego. La gráfica a continuación muestra las puntuaciones de cada jugador y la valoración media recibida.

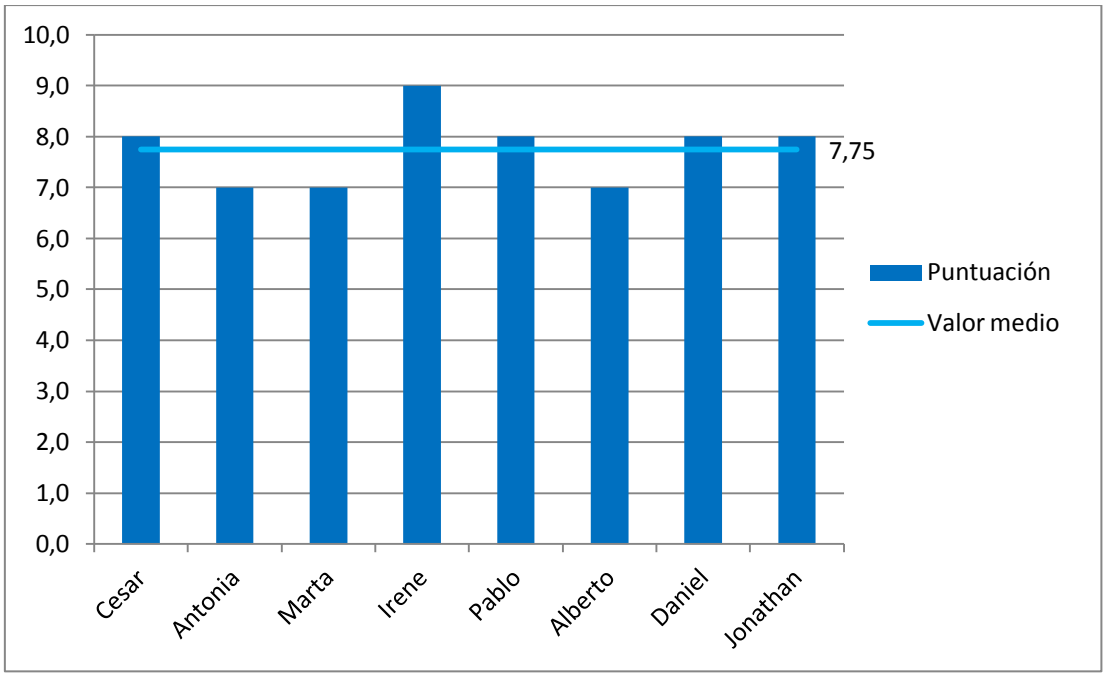


Figura 6.7. Valoración general. Resultados.

Como se puede observar, ningún encuestado ha valorado el juego por debajo de ‘7’ y la puntuacion media es de ‘7.75’. El resultado de este apartado es bastante favorable e indica un alto grado de satisfacción general de los jugadores respecto al juego, a pesar de que se puede mejorar en algunos aspectos y detalles.

Se pueden ver todos los test realizados en el Anexo B.

6.4. Conclusiones

Tras la realización de la fase de pruebas, se puede concluir que en base al estudio realizado se ha conseguido:

- Determinar los puntos fuertes y débiles del proyecto, de manera que se tienen detectados los aspectos en los que se debe trabajar en el futuro.

- Comprobar que el juego se puede enfocar hacia cualquier tipo de persona independientemente de su género o edad, ya que observando los resultados obtenidos no hay un patrón que nos haga descartar a ningún tipo de persona.

- Saber que detalles del juego gustan y no gustan a los jugadores y por lo tanto saber cuales están bien escogidos y cuales se deben mejorar.

- Conocer de primera mano la opinión general del juego por parte de los encuestados.

APÉNDICE

A

Planificación y presupuesto

Es necesaria una planificación previa al proyecto que sirva para estructurarlo en varias fases y asignar a cada una de ellas los recursos necesarios (materiales, coste, personas, tiempo). Esta planificación ayuda a verificar el cumplimiento de plazos y funcionalidades previstas.

En este apartado se muestra el ciclo de vida de un juego comercial, la planificación y etapas seguidas en la elaboración de este proyecto, el presupuesto elaborado, como se comercializa un videojuego en Android Market y la planificación final del proyecto.

A.1. Ciclo de vida de un videojuego comercial

En este apartado se detalla el ciclo de vida usual de un videojuego comercial desde la primera idea elegida hasta la última actualización añadida. La lista de etapas no se ha seguido rigurosamente aunque es aplicable a la planificación de cualquier juego.

- **Concepto o idea:** En esta fase se define el concepto de juego que se quiere desarrollar y tras una tormenta de ideas se crean la propuesta del juego y el arte conceptual del mismo. Estos elementos sirven para presentar inicialmente el juego.

- **Pre-producción:** En esta etapa se valora si realmente merece la pena desarrollar el proyecto y se diseña la mecánica del juego (niveles, personajes, enemigos, etc.). Debe incluir además un cronograma inicial de implementación.

- **Producción:** Se trata de la fase más importante del ciclo de vida ya que en ella se debe realizar la implementación de la aplicación. Se compone de varias etapas como la programación, la creación del audio y diferentes niveles, creación del arte grafico y fase de testeo. En esta última fase los testeadores colaboran con

los desarrolladores para solucionar los errores que surjan y añadir o suprimir funcionalidades.

- **Alfa:** En esta etapa el juego ya es jugable de principio a fin pero pueden quedar por depurar aún muchos detalles o errores por solucionar.

- **Beta:** En esta fase los diferentes módulos del juego deben estar terminados y añadidos, y únicamente debe ser necesario depurar los errores para poder liberar el juego.

- **Congelación del código:** En esta fase el código se congela y se han debido solucionar los errores detectados en la fase beta (al menos los más importantes). Se encuentra pendiente de confirmación antes de convertirse en la versión final.

- **Liberación:** El juego ha sido aprobado y se envía a los canales de distribución, en formato físico y/o digital, para que los usuarios puedan acceder a él.

- **Parches:** En muchas ocasiones, quizás debido a las prisas o limitaciones en el tiempo, se vuelve inevitable el uso de parches que solucionen errores detectados después del lanzamiento del juego.

- **Actualizaciones:** Representa contenido adicional que se quiere añadir al juego para mejorarlo. Las actualizaciones pueden ir desde mínimos detalles hasta grandes mejoras como nuevos niveles, personajes, enemigos, etc.

A.2. Planificación y etapas del proyecto

Previo a la planificación, es necesario decidir qué modelo de ciclo de vida se va a utilizar en el desarrollo de la aplicación. De entre los diferentes modelos (lineal, cascada, sashimi, etc.) se ha elegido el modelo evolutivo.

Este modelo es ideal para el proyecto ya que ofrece gran flexibilidad para la inclusión de nuevos requerimientos, ya que es muy complicado obtenerlos todos al inicio, permitiendo añadir o mejorar la funcionalidad de los diferentes módulos del proyecto en cualquier momento. Como muestra la figura A.1 este modelo se basa en la iteración del ciclo requerimientos-desarrollo-evaluación

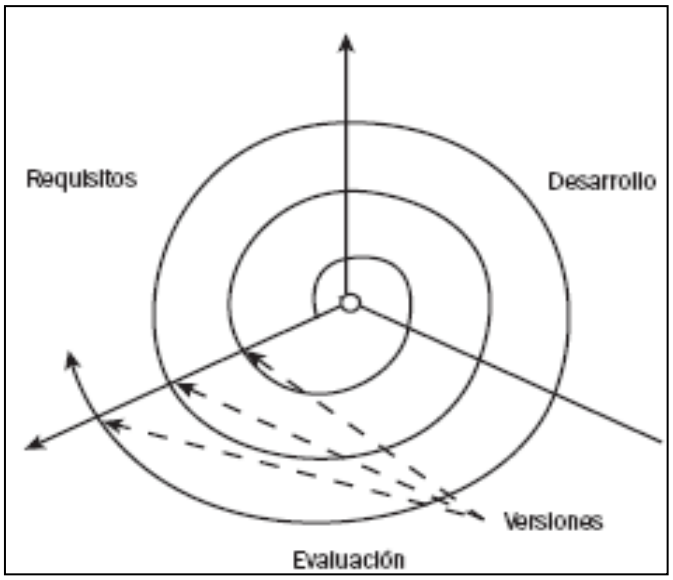


Figura A.1. Ciclo de vida evolutivo. Diagrama.

Finalizada la fase de desarrollo se tiene una versión del producto que es evaluada, tras la cual se pueden añadir requerimientos nuevos o revisar los actuales si la evaluación no es satisfactoria, consiguiendo que el producto se pueda ir puliendo y mejorando continuamente.

En los diferentes módulos o componentes del proyecto también se ha aplicado este modelo de manera que ha sido posible depurarlos por separado hasta obtener la versión final.

Los componentes son:

- **Formación inicial:** Debe familiarizarse con los elementos necesarios para el desarrollo del proyecto y estudiar tanto el entorno de desarrollo como con la plataforma para la que se implementa la aplicación.

- **Memoria y documentación:** A medida que se avanza en el desarrollo del proyecto es necesario recoger toda la documentación necesaria (análisis, diseño, problemas encontrados, variaciones, etc.) para redactar el documento final.

- **Aspectos y estructura básica:** Este módulo aporta la funcionalidad básica para que la aplicación funcione. Incluye inicio y fin de la aplicación, detección e implementación de eventos (pantalla, botones) y el paso entre actividades.

- **Interfaz gráfico:** Se diseñan las apariencias de las diferentes pantallas y se crean los diferentes menús, diálogos e iconos.

- **Núcleo del juego:** Se trata del componente más extenso ya que engloba toda la gestión de movimientos y disparo, colisiones y el ciclo de vida de la aplicación.

- **Otros:** En este componente se incluyen los sonidos y la vibración.

- **Evaluación y pruebas:** Una vez obtenida la versión completa se debe evaluar y comprobar que todo funciona según lo esperado.

Estos componentes se muestran en las tablas y diagramas de planificación, donde se indican los recursos y tiempos que han sido necesarios utilizar.

1. Formación inicial	20 días
1.1 Estudio Eclipse	5 días
1.2 Estudio Android	20 días
2. Memoria y documentación	140 días
3. Fase 1: Aspectos y estructura básica	11 días
3.1 Requisitos de fase 1	4 días
3.1.1 Navegación entre actividades	3 días
3.1.2 Eventos básicos	2 días
3.1.3 Arranque y salida del juego	4 días
3.2 Diseño	2 días
3.3 Implementación	4 días
3.4 Evaluación	1 día
4. Fase 2: Interfaz gráfico	17 días
4.1 Requisitos de fase 2	4 días
4.1.1 Diseño de pantallas	4 días
4.1.2 Menús	2 días
4.1.3 Cuadros de diálogo	1 día
4.1.4 Iconos	1 día
4.2 Diseño	3 días
4.3 Implementación	8 días
4.4 Evaluación	2 días
5. Fase 3: Núcleo del juego	39 días
5.1 Requisitos de fase 3	15 días
5.1.1 Gestión de movimientos	15 días
5.1.2 Gestión de disparo	8 días

5.1.3 Colisiones	6 días
5.1.4 Ciclo de vida del juego	6 días
5.2 Diseño	7 días
5.3 Implementación	12 días
5.4 Evaluación	5 días
6. Fase 4: Otros componentes	7 días
6.1 Requisitos de fase 4	4 días
6.1.1 Sonidos	4 días
6.1.2 Vibración	1 día
6.2 Diseño	1 días
6.3 Implementación	1 días
6.4 Evaluación	1 día
7. Evaluación y últimas pruebas	6 días

Tabla A.1. Planificación temporal del proyecto.

Todas las tareas de la tabla A.1 requieren de un recurso material o personal. En los módulos o componentes donde se encuentran disponibles más de un recurso es posible llevar a cabo varias tareas en paralelo, lo cual optimiza el tiempo empleado en su realización. También es importante realizar reuniones periódicas para mantener un seguimiento del proyecto.

En la figura A.2 se muestra la planificación inicial de las diferentes tareas con sus fechas de inicio y fin y los recursos asignados a cada una. A pesar de que inicialmente se indican varios tipos de recursos humanos (artista conceptual, analista, tester, desarrollador) en la práctica solamente ha participado una persona, el autor del proyecto.

Las figuras A.3 y A.4 muestran la evolución temporal de las diferentes tareas sin tener en cuenta los sábados ni los domingos. La planificación del proyecto ha establecido la duración en seis meses y medio, desde Marzo de 2011 hasta mediados de Septiembre de 2011.

Diagrama de Gantt

	Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos
1	**Formacion inicial**	**20 días**	**mar 01/03/11**	**lun 28/03/11**	**Desarrollador**
2	Estudio Eclipse	5 días	mar 01/03/11	lun 07/03/11	Desarrollador
3	Estudio Android	20 días	mar 01/03/11	lun 28/03/11	Desarrollador
4	Memoria y documentacion	140 días	mar 01/03/11	lun 12/09/11	Desarrollador
5	**Fase 1: Aspectos y estructura básica**	**11 días**	**mar 29/03/11**	**mar 12/04/11**	
6	**Requisitos de fase 1**	**4 días**	**mar 29/03/11**	**vie 01/04/11**	
7	Navegación entre actividades	3 días	mar 29/03/11	jue 31/03/11	Analista
8	Eventos básicos	2 días	mar 29/03/11	mié 30/03/11	Desarrollador
9	Arranque y salida del juego	4 días	mar 29/03/11	vie 01/04/11	Analista
10	Diseño	2 días	lun 04/04/11	mar 05/04/11	Analista
11	Implementacion	4 días	mié 06/04/11	lun 11/04/11	Desarrollador
12	Evaluacion	1 día	mar 12/04/11	mar 12/04/11	Tester
13	**Fase 2: Interfaz Gráfico**	**17 días**	**mié 13/04/11**	**jue 05/05/11**	
14	**Requisitos de fase 2**	**4 días**	**mié 13/04/11**	**lun 18/04/11**	
15	Diseño de pantallas	4 días	mié 13/04/11	lun 18/04/11	Artista conceptual
16	Menús	2 días	mié 13/04/11	jue 14/04/11	Desarrollador
17	Cuadros de diálogo	1 día	mié 13/04/11	mié 13/04/11	Desarrollador
18	Iconos	1 día	mié 13/04/11	mié 13/04/11	Artista conceptual
19	Diseño	3 días	mar 19/04/11	jue 21/04/11	Analista
20	Implementacion	8 días	vie 22/04/11	mar 03/05/11	Desarrollador
21	Evaluacion	2 días	mié 04/05/11	jue 05/05/11	Tester
22	**Fase 3: Nucleo del juego**	**39 días**	**vie 06/05/11**	**mié 29/06/11**	
23	**Requisitos de fase 3**	**15 días**	**vie 06/05/11**	**jue 26/05/11**	
24	Gestion de movimientos	15 días	vie 06/05/11	jue 26/05/11	Desarrollador
25	Gestion de disparo	8 días	vie 06/05/11	mar 17/05/11	Desarrollador
26	Colisiones	6 días	vie 06/05/11	vie 13/05/11	Desarrollador
27	Ciclo de vida del juego	6 días	vie 06/05/11	vie 13/05/11	Desarrollador
28	Diseño	7 días	vie 27/05/11	lun 06/06/11	Analista
29	Implementación	12 días	mar 07/06/11	mié 22/06/11	Desarrollador
30	Evaluación	5 días	jue 23/06/11	mié 29/06/11	Tester
31	**Fase 4: Otros componentes**	**7 días**	**jue 30/06/11**	**vie 08/07/11**	
32	**Requisitos de fase 4**	**4 días**	**jue 30/06/11**	**mar 05/07/11**	
33	Sonidos	4 días	jue 30/06/11	mar 05/07/11	Desarrollador
34	Vibración	1 día	jue 30/06/11	jue 30/06/11	Desarrollador
35	Diseño	1 día	mié 06/07/11	mié 06/07/11	Analista
36	Implementación	1 día	jue 07/07/11	jue 07/07/11	Desarrollador
37	Evaluacion	1 día	vie 08/07/11	vie 08/07/11	Tester
38	Evaluacion y últimas pruebas	6 días	lun 11/07/11	lun 18/07/11	

Figura A.2. Diagrama de Gantt. Tareas y recursos.

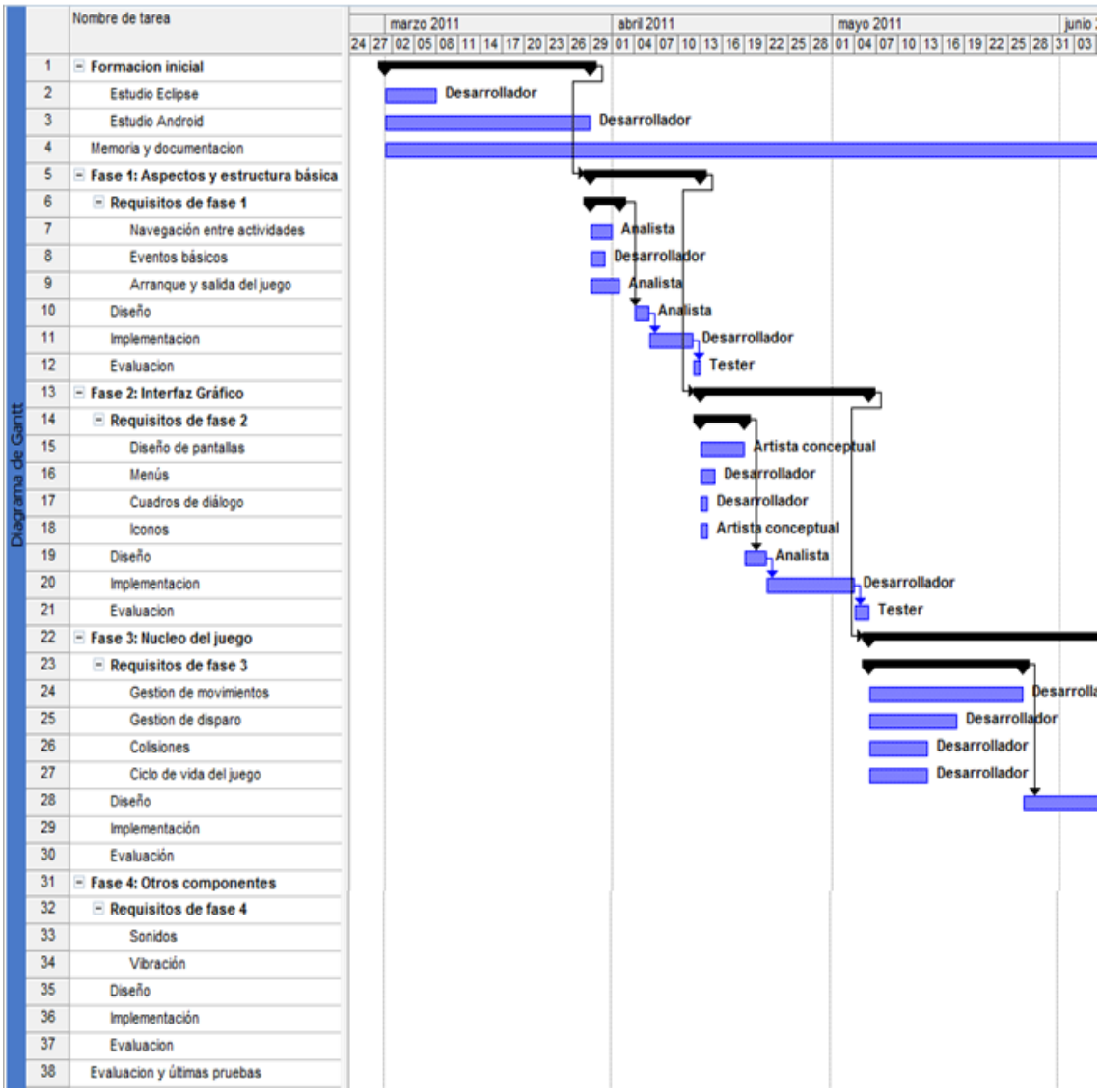


Figura A.3. Diagrama de Gantt. Evolución temporal I.

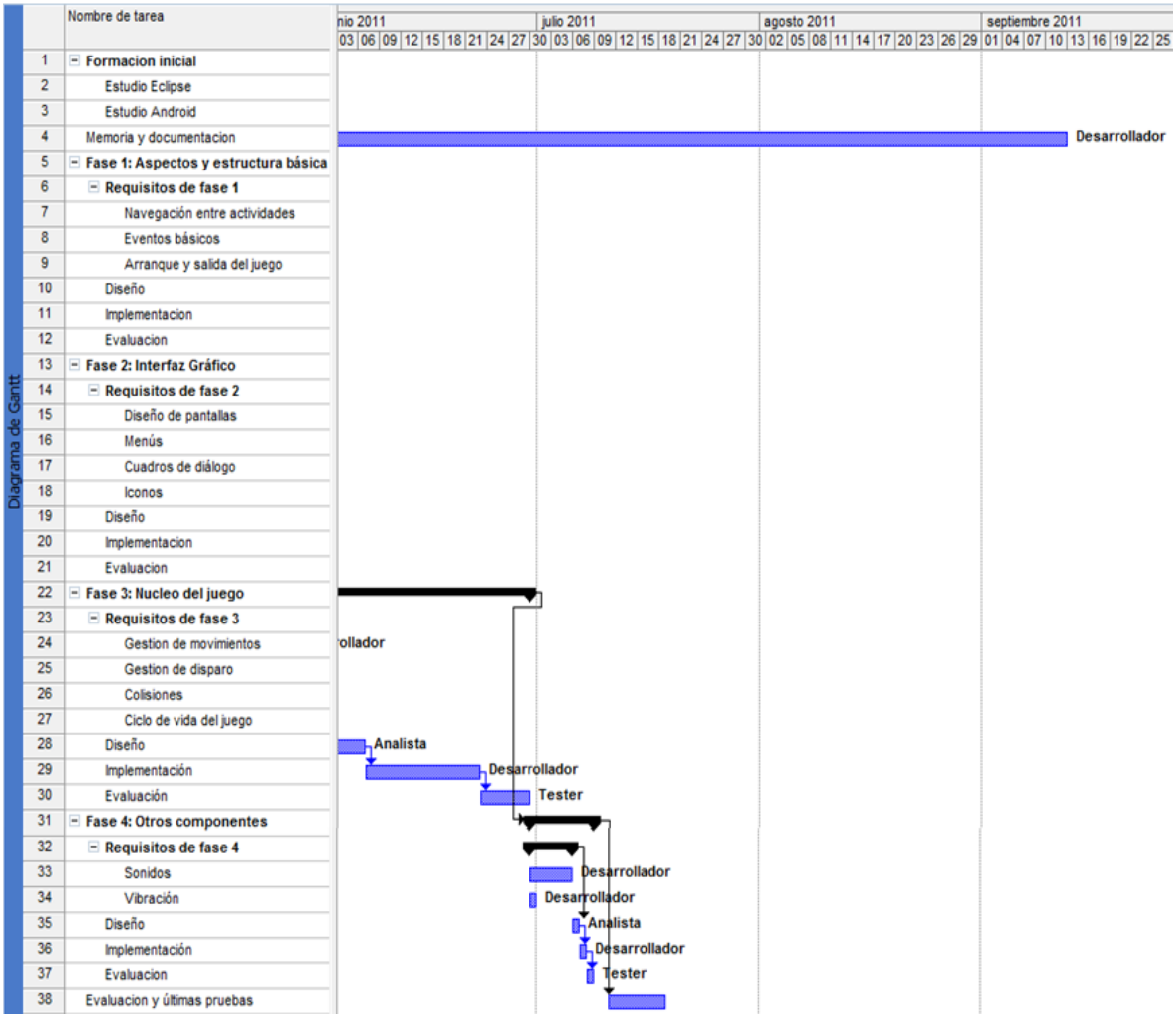


Figura A.4. Diagrama de Gantt. Evolución temporal II.

A.3. Presupuesto del proyecto

En este apartado se detallan los costes asociados al proyecto (figuras A.5 y A.6). En él se incluyen tanto costes directos (personal, materiales, etc.) como indirectos (viajes, dietas, etc.).

El presupuesto total es de 12.391,19 euros repartidos en varias partidas, la más costosa es la de personal con un gasto de 10.650 euros. También es importante tener en cuenta los costes asociados a los equipos utilizados (ordenador y dispositivo smartphone) así como los desplazamientos realizados para mantener reuniones periódicas.

UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITECNICA SUPERIOR

PRESUPUESTO DEL PROYECTO

1 - Autor

Pablo Covarrubias Herrera

2 - Departamento

Departamento de informática

3 - Descripción del proyecto

· Título: Diseño e implementación del juego Super Pang para plataforma Android

· Duración (meses): 6

· Tasa de costes indirectos: 15%

4 - Presupuesto total del proyecto

12.391,19 Euros

5 - Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	NIF	Categoría	Dedicación (meses)	Costes (mes)	Costes Totales	Firma de conformidad
Peralta Donate, Juan		Ingeniero Superior	1	1.900,00	1.900,00	
Covarrubias Herrera, Pablo		Ingeniero Junior	7	1.250,00	8.750,00	
				Total	10.650,00	

EQUIPOS

Descripción	Coste (euro)	% uso dedicado a proyecto	Dedicación (meses)	Periodo de deprecación	Coste imputable [a]
Ordenador	700,00	100	6	60	70,00
HTC Wildfire	49,95	100	6	60	4,95
				Total	74,95

Figura A.5. Presupuesto I.

[a] Fórmula de cálculo de la amortización

$$\frac{A}{B} xCxD$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = Periodo de deprecación (60 meses)
C = Coste del equipo (sin IVA)
D = % de uso dedicado a proyecto (100% normalmente)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste computable
	Total	

COSTES INDIRECTOS DEL PROYECTO [b]

Descripción	Empresa	Coste computable
Viajes		50,00
	Total	50,00

[b] Incluidos todos los gastos no indicados anteriormente: consumibles, viajes, dietas,etc.

6 - Resumen presupuestario

Concepto costes totales	Presupuesto costes totales
Personal	10.650,00
Amortización	74,95
Tareas subcontratadas	0,00
Costes de funcionamiento	50,00
Costes indirectos	1.616,24
Total	12.391,19

Figura A.6. Presupuesto II.

A.4. Comercialización de la aplicación

Tras realizar la planificación del juego y la elaboración del presupuesto, es necesario evaluar el mercado para determinar si nuestro juego será rentable o no. El mercado evaluado debe ser el Android Market ya que es la plataforma para la que se ha implementado el juego [24].

Dicho mercado permite a los desarrolladores publicar sus aplicaciones para su distribución a los usuarios de Android. Es necesario registrarse como desarrollador, lo cual permite tener el control de la disponibilidad de sus aplicaciones a los usuarios. El desarrollador puede visualizar todas sus aplicaciones incluyendo sus puntuaciones, numero de descargas y comentarios. Además es libre de actualizar o añadir nuevas versiones de su aplicación cuando lo desee.

Para el registro como desarrollador el usuario debe realizar tres pasos: crear un perfil de desarrollador, pagar una cuota de registro de 25 dólares y aceptar el acuerdo de distribución para desarrolladores de Android Market.

Una vez registrado, el desarrollador puede poner precio a sus aplicaciones o dejarlas accesibles a los usuarios sin coste alguno. Para las aplicaciones de pago, Google establece una tarifa de transacción del 30% del precio de la aplicación de manera que el desarrollador se queda con el restante 70%. Estableciendo 1,00 euros como precio de descarga de la aplicación, por cada copia vendida el beneficio neto será de 0,70 euros.

Cuándo la aplicación se encuentre disponible en el Market, es posible disponer de una ubicación destacada, mediante opciones que facilitan la búsqueda de aplicaciones en forma de listas, algunas de ellas elegidas por el equipo de Android Market. Además no existen anuncios publicitarios ni espacios promocionales de pago en el Market.

Incluso con aplicaciones gratuitas es posible obtener ingresos. Para ello se utiliza publicidad insertada en la aplicación de manera que cuando el usuario la selecciona, el desarrollador se lleva una bonificación. Para este proyecto no se ha tenido en cuenta esta opción ya que se ha considerado suficiente el ingreso por descarga.

Tras el análisis del mercado de aplicaciones, se ha evaluado la amortización del presupuesto diseñado. Vendiendo la aplicación a 1,00 euros, y por lo tanto obteniendo 0,70 euros de beneficio, se calcula que para amortizar los 12.391,19 euros del coste del proyecto es necesario vender 17.702 copias, lo que supone una media de 1.475 descargas mensuales en un año. Si bien es cierto que las aplicaciones con mayor éxito de descargas son gratuitas, esta cifra se considera asequible tras analizar los juegos similares en el Market.

A.5. Planificación final

Durante el desarrollo del proyecto la planificación inicial sufrió algunas variaciones. A continuación se muestran dichas variaciones y la planificación final del proyecto (figura A.7).

Las tareas que sufrieron modificación fueron las siguientes:

- **Memoria y documentación:** En lugar de comenzar el desarrollo del proyecto y la memoria al mismo tiempo, se comenzó el documento con el desarrollo ya iniciado.

- **Fase 1: Aspectos y estructura básica:** Esta fase inicialmente tenía asignado 11 días pero surgieron contratiempos al no estar familiarizado con la plataforma para la que se implementaba que aumentaron su duración hasta los 16 días.

Navegación entre actividades: Para navegar entre actividades era necesario manejar los eventos básicos de los elementos de las actividades con lo cual esta tarea se realizo posterior a la tarea *eventos básicos* en lugar de en paralelo como estaba programada. Además se necesito invertir un día más del programado inicialmente.

Arranque y salida del juego: Fue necesario invertir un día menos del planificado en esta tarea, aunque esta reducción no significó variación del tiempo

Diseño, implementación y evaluación: Estas tareas, debido a las mencionadas modificaciones, sufrieron un aumento de duración de un día

- **Fase 2: Interfaz gráfico:** Esta fase aumentó su duración en 4 días respecto a la previsión inicial, requiriendo 21 días en lugar de 17.

Diseño de pantallas: Inicialmente no se tenía práctica con la creación de interfaces gráficos y fue necesario invertir dos días adicionales a la planificación prevista

Diseño e implementación: Debido a la anterior modificación, ambas tareas aumentaron un día su duración.

- **Fase 3: Núcleo del juego:** Es la fase que mas contratiempos tuvo. Se incrementó su duración de los 39 días iniciales a 71.

Gestión de movimientos: Aumentó en 5 días su duración ya que se tuvo que modificar el movimiento previsto de las pelotas del juego.

Gestión de disparo, colisiones y ciclo de vida: No sufrieron variación en su duración, pero el contratiempo anterior provoco que no se quisiera precipitar en el desarrollo de las diferentes tareas y se realizaron una tras otra, en lugar de en paralelo como estaba planificado inicialmente.

Diseño y evaluación: Ambas tareas necesitaron de un día más del previsto.

Implementación: Debido al aumento de complejidad en la gestión de movimientos, esta tarea incrementó su duración en 5 días.

- **Evaluación y últimas pruebas:** La variación en las diferentes funcionalidades provocaron que se necesitaran 4 días más de los previstos para analizar y probar la funcionalidad completa del juego.

Además de los problemas indicados, no se tuvo en cuenta inicialmente que tan solo se contaba con un ingeniero por lo que la duración se incremento debido a las vacaciones del mes de agosto.

Diagrama de Gantt

	Nombre de tarea	Duración	Comienzo	Fin
1	**Formacion inicial**	**20 días**	**mar 01/03/11**	**lun 28/03/11**
2	Estudio Eclipse	5 días	mar 01/03/11	lun 07/03/11
3	Estudio Android	20 días	mar 01/03/11	lun 28/03/11
4	Memoria y documentacion	140 días	mié 20/04/11	vie 02/12/11
5	**Fase 1: Aspectos y estructura básica**	**16 días**	**mar 29/03/11**	**mar 19/04/11**
6	**Requisitos de fase 1**	**6 días**	**mar 29/03/11**	**mar 05/04/11**
7	Navegación entre actividades	4 días	jue 31/03/11	mar 05/04/11
8	Eventos básicos	2 días	mar 29/03/11	mié 30/03/11
9	Arranque y salida del juego	3 días	mar 29/03/11	jue 31/03/11
10	Diseño	3 días	mié 06/04/11	vie 08/04/11
11	Implementacion	5 días	lun 11/04/11	vie 15/04/11
12	Evaluacion	2 días	lun 18/04/11	mar 19/04/11
13	**Fase 2: Interfaz Gráfico**	**21 días**	**mié 20/04/11**	**mié 18/05/11**
14	**Requisitos de fase 2**	**6 días**	**mié 20/04/11**	**mié 27/04/11**
15	Diseño de pantallas	6 días	mié 20/04/11	mié 27/04/11
16	Menús	2 días	mié 20/04/11	jue 21/04/11
17	Cuadros de diálogo	1 día	mié 20/04/11	mié 20/04/11
18	Iconos	1 día	mié 20/04/11	mié 20/04/11
19	Diseño	4 días	jue 28/04/11	mar 03/05/11
20	Implementacion	9 días	mié 04/05/11	lun 16/05/11
21	Evaluacion	2 días	mar 17/05/11	mié 18/05/11
22	**Fase 3: Nucleo del juego**	**71 días**	**jue 19/05/11**	**mar 27/09/11**
23	**Requisitos de fase 3**	**40 días**	**jue 19/05/11**	**mié 13/07/11**
24	Gestion de movimientos	20 días	jue 19/05/11	mié 15/06/11
25	Gestion de disparo	8 días	jue 16/06/11	lun 27/06/11
26	Colisiones	6 días	mar 28/06/11	mar 05/07/11
27	Ciclo de vida del juego	6 días	mié 06/07/11	mié 13/07/11
28	Diseño	8 días	jue 14/07/11	lun 25/07/11
29	Implementación	17 días	mar 26/07/11	lun 19/09/11
30	Evaluación	6 días	mar 20/09/11	mar 27/09/11
31	**Fase 4: Otros componentes**	**7 días**	**mié 28/09/11**	**jue 06/10/11**
32	**Requisitos de fase 4**	**4 días**	**mié 28/09/11**	**lun 03/10/11**
33	Sonidos	4 días	mié 28/09/11	lun 03/10/11
34	Vibración	1 día	mié 28/09/11	mié 28/09/11
35	Diseño	1 día	mar 04/10/11	mar 04/10/11
36	Implementación	1 día	mié 05/10/11	mié 05/10/11
37	Evaluacion	1 día	jue 06/10/11	jue 06/10/11
38	Evaluacion y últimas pruebas	10 días	vie 07/10/11	jue 20/10/11

Figura A.7. Planificación final. Diagrama de Gantt.

B

Test de jugabilidad

TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:

◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:

◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

2. El juego te ha parecido fácil en general:

◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

EFECTIVIDAD

1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):

◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:

◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

2. La consideras la interfaz gráfica bien diseñada:

◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

3. Te parece agradable la interfaz gráfica:

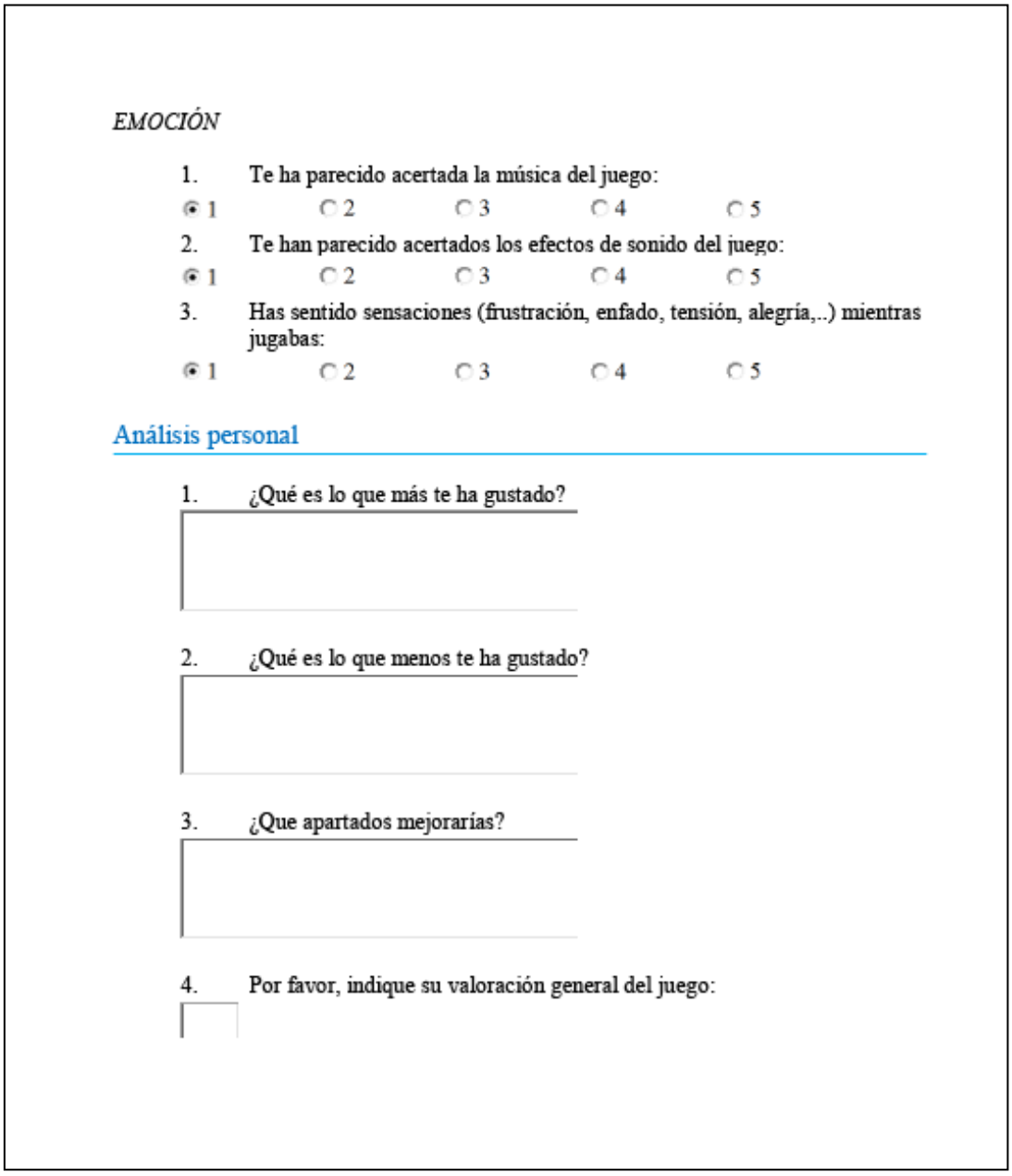
◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:

◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

Figura B.1. Encuesta. Parte I.



EMOCIÓN

1. Te ha parecido acertada la música del juego:

(•) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

2. Te han parecido acertados los efectos de sonido del juego:

(•) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:

(•) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?

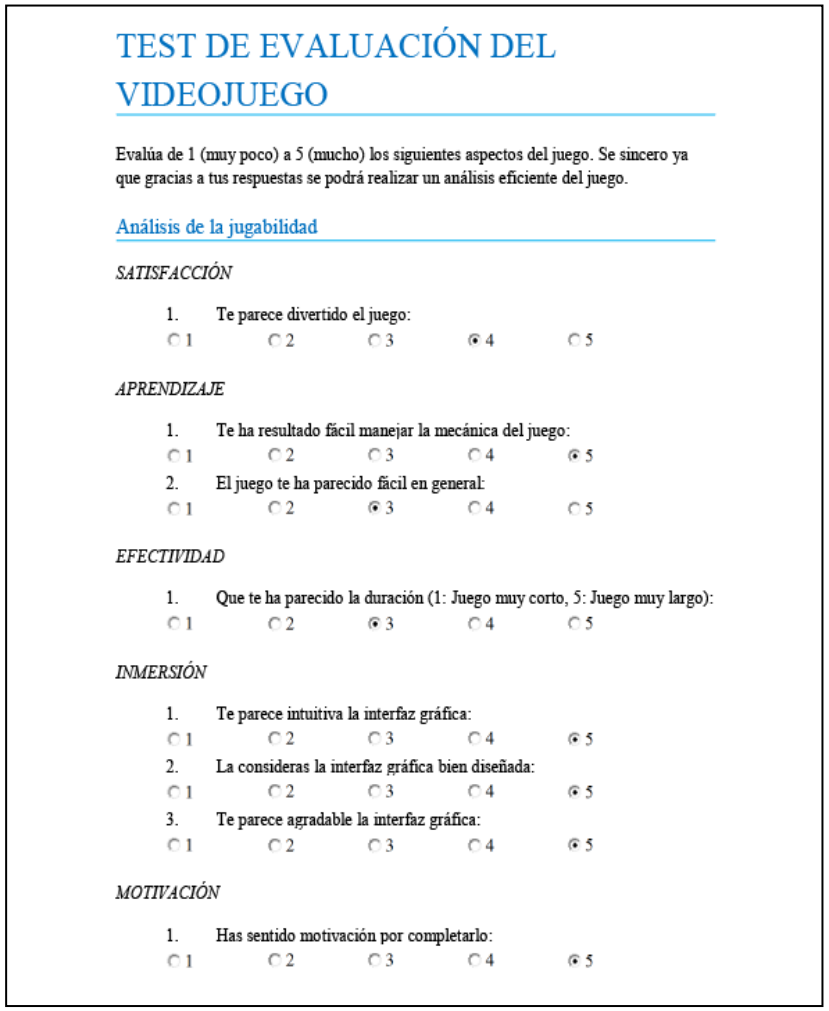
2. ¿Qué es lo que menos te ha gustado?

3. ¿Que apartados mejorarías?

4. Por favor, indique su valoración general del juego:

Figura B.2. Encuesta. Parte II.

Marta Covarrubias Herrera, 20 años.



TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. El juego te ha parecido fácil en general:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

EFECTIVIDAD

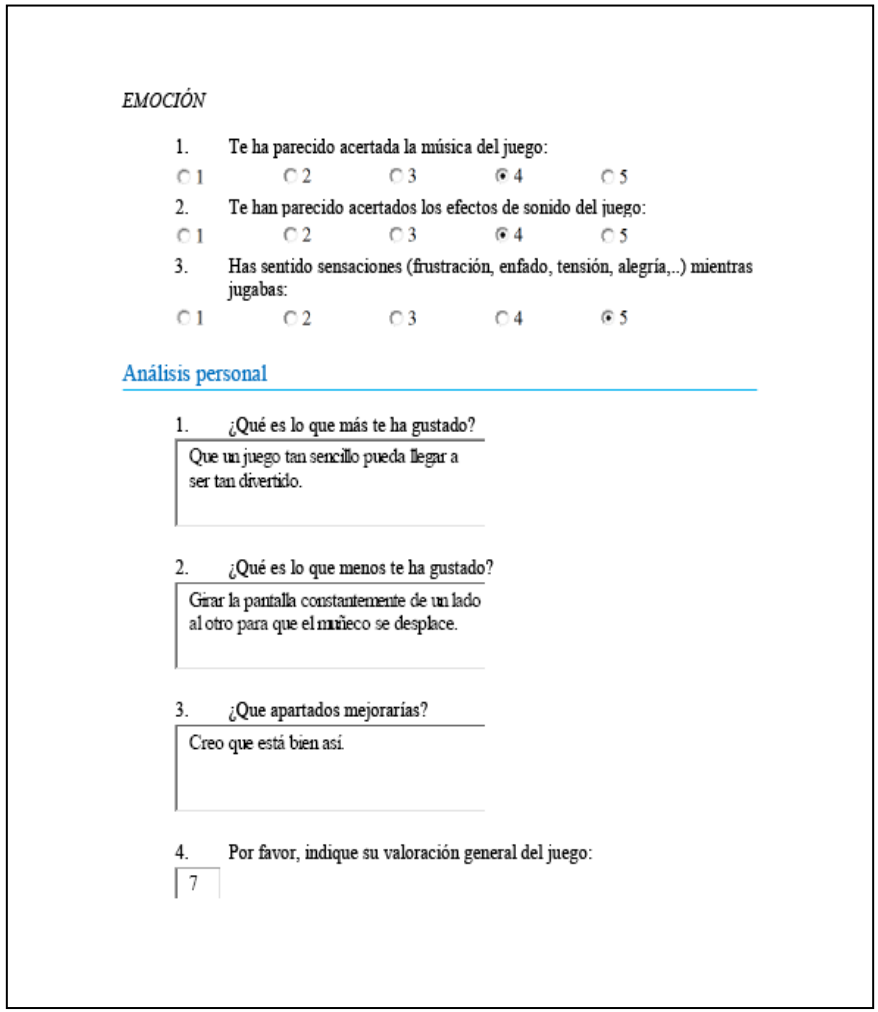
1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. La consideras la interfaz gráfica bien diseñada:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
3. Te parece agradable la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5



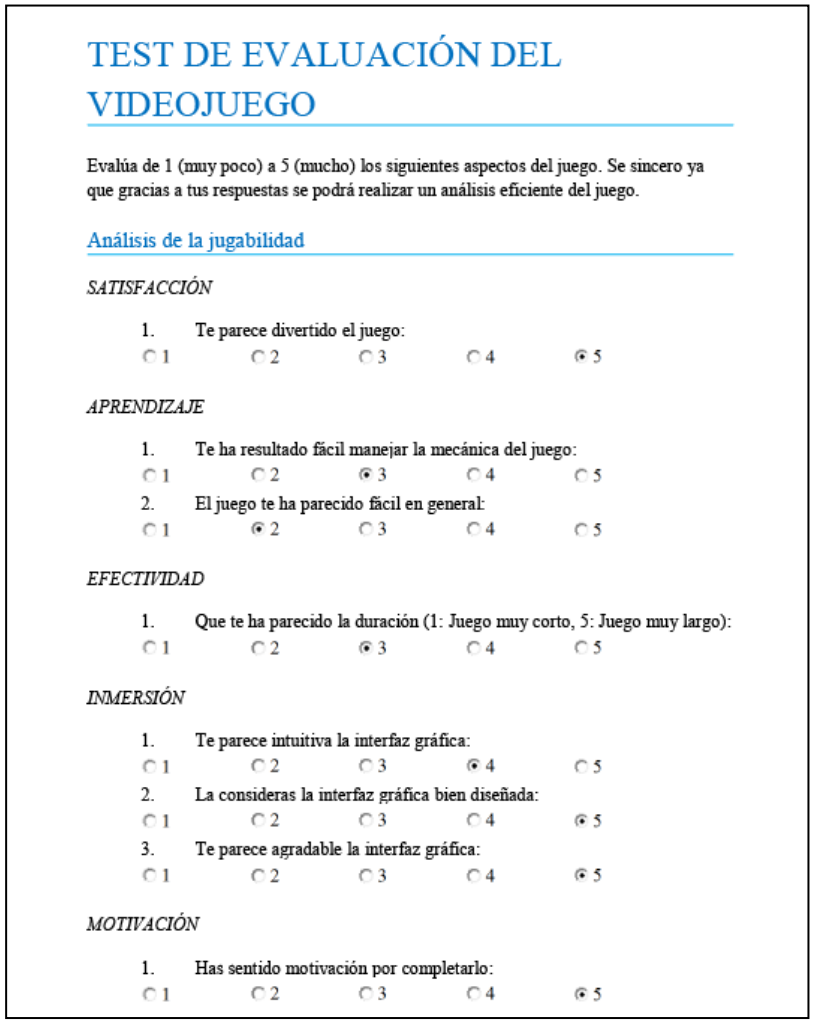
EMOCIÓN

1. Te ha parecido acertada la música del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. Te han parecido acertados los efectos de sonido del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?
   Que un juego tan sencillo pueda llegar a ser tan divertido.
2. ¿Qué es lo que menos te ha gustado?
   Girar la pantalla constantemente de un lado al otro para que el muñeco se desplace.
3. ¿Que apartados mejorarías?
   Creo que está bien así.
4. Por favor, indique su valoración general del juego:
   7

César Juárez Mejias, 24 años.



TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5
2. El juego te ha parecido fácil en general:
   ○ 1 ◉ 2 ○ 3 ○ 4 ○ 5

EFECTIVIDAD

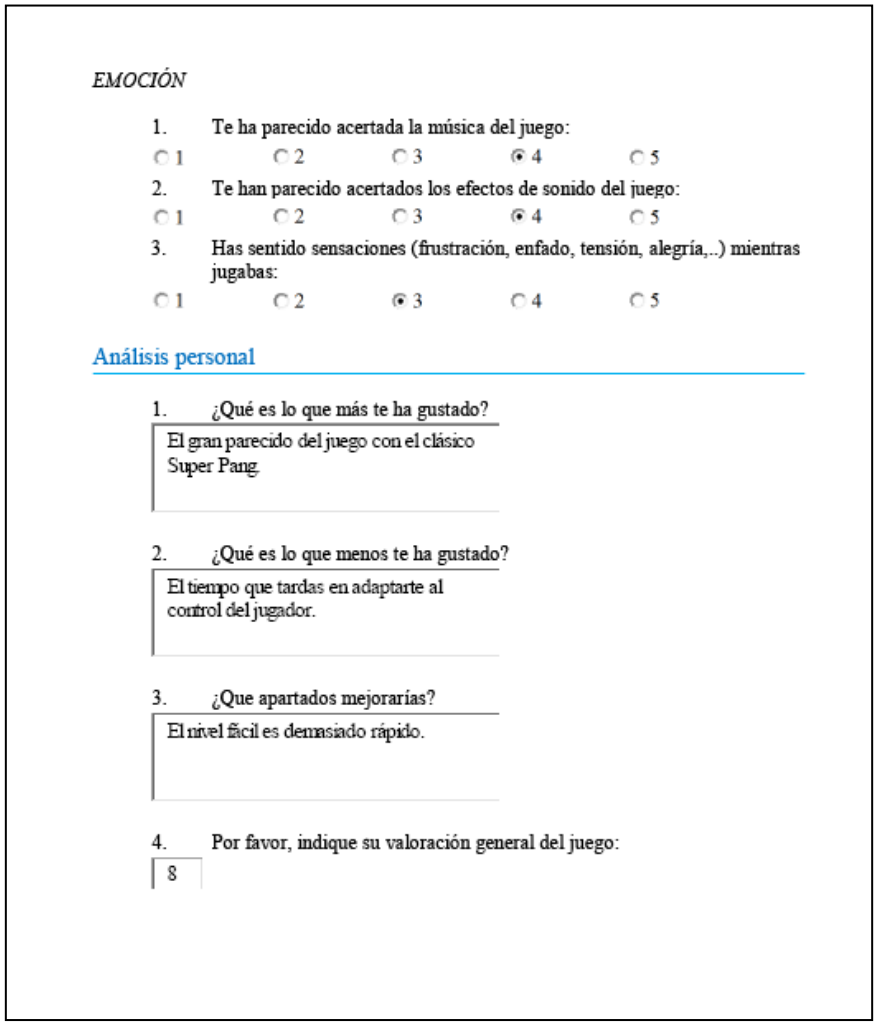
1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. La consideras la interfaz gráfica bien diseñada:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
3. Te parece agradable la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5



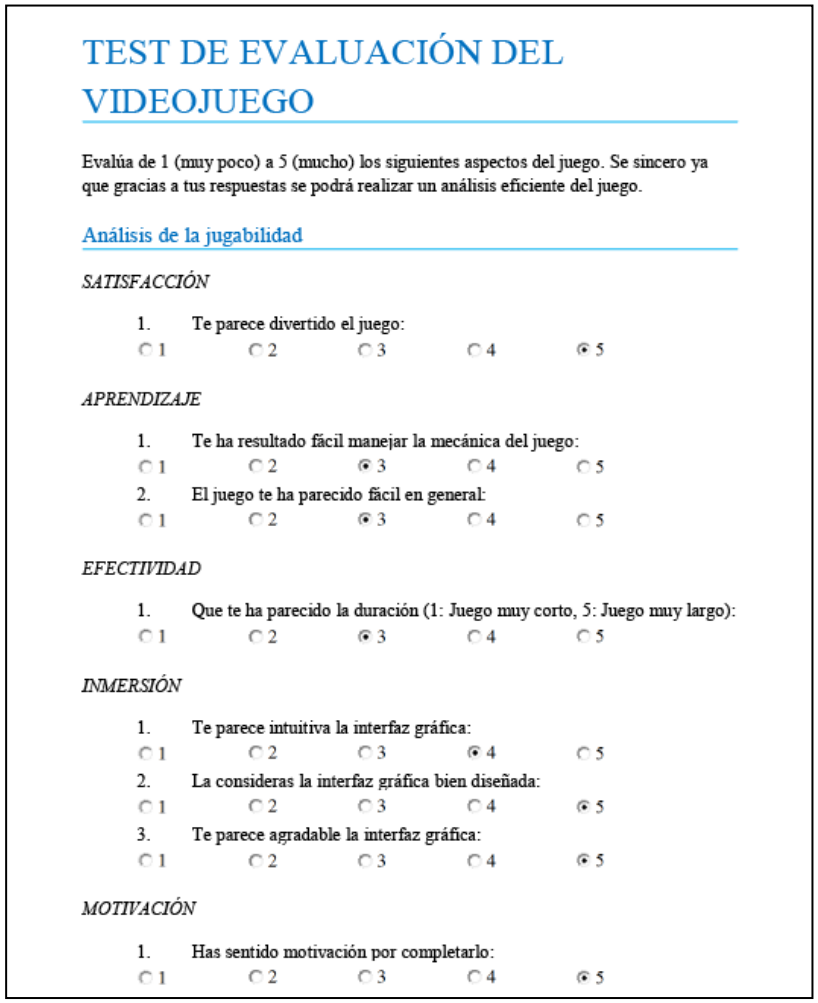
EMOCIÓN

1. Te ha parecido acertada la música del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. Te han parecido acertados los efectos de sonido del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?
   El gran parecido del juego con el clásico Super Pang.
2. ¿Qué es lo que menos te ha gustado?
   El tiempo que tardas en adaptarte al control del jugador.
3. ¿Que apartados mejorarías?
   El nivel fácil es demasiado rápido.
4. Por favor, indique su valoración general del juego:
   8

Antonia Herrera Rodriguez, 52 años.



TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5
2. El juego te ha parecido fácil en general:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

EFECTIVIDAD

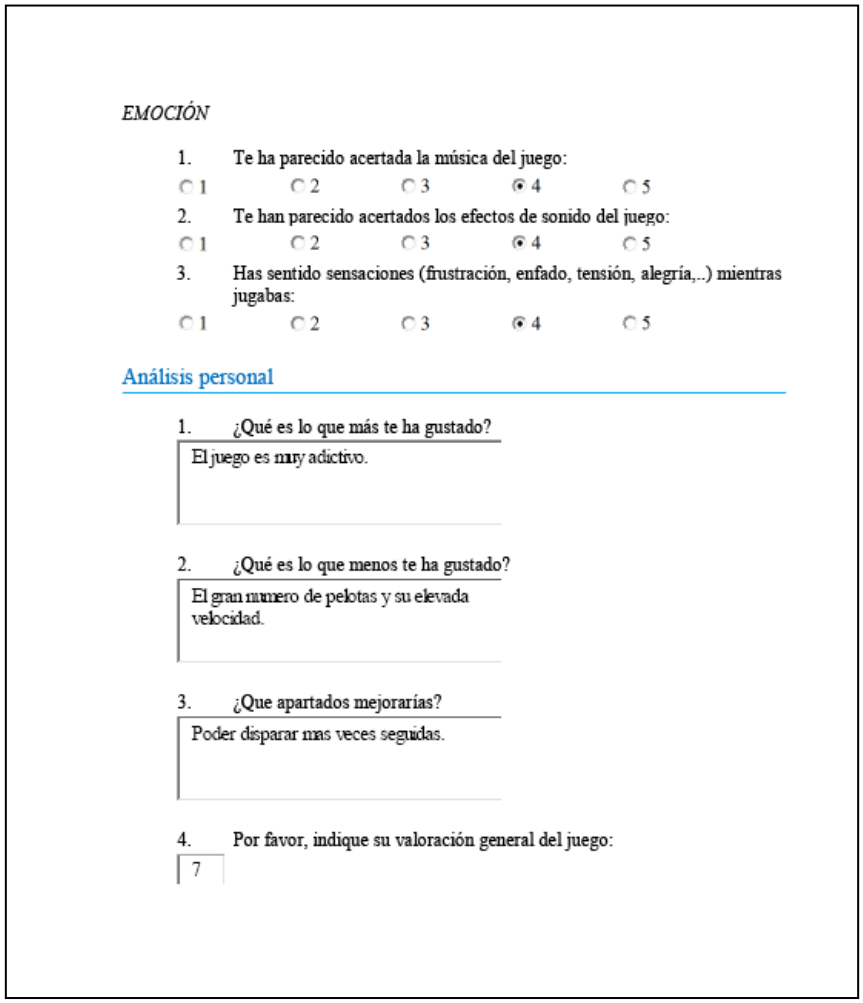
1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. La consideras la interfaz gráfica bien diseñada:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
3. Te parece agradable la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5



EMOCIÓN

1. Te ha parecido acertada la música del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. Te han parecido acertados los efectos de sonido del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?
   El juego es muy adictivo.
2. ¿Qué es lo que menos te ha gustado?
   El gran numero de pelotas y su elevada velocidad.
3. ¿Que apartados mejorarías?
   Poder disparar mas veces seguidas.
4. Por favor, indique su valoración general del juego:
   7

Pablo Covarrubias Burguillo, 57 años.

TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. El juego te ha parecido fácil en general:
   ◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

EFECTIVIDAD

1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. La consideras la interfaz gráfica bien diseñada:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
3. Te parece agradable la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

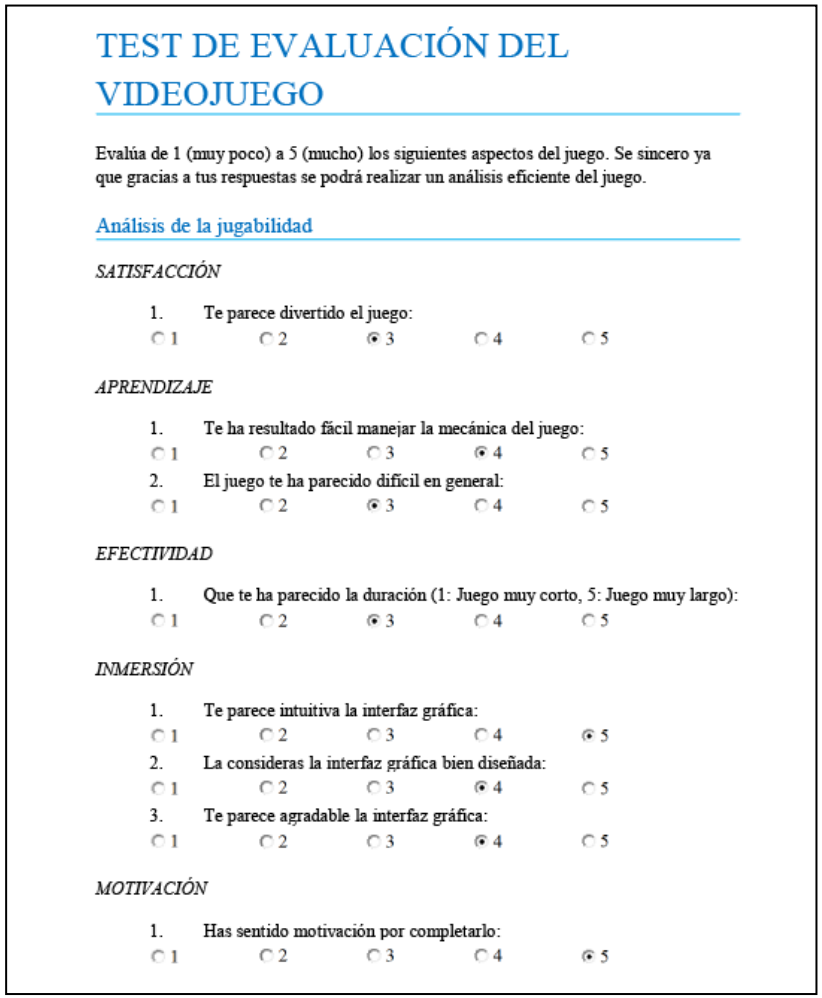
EMOCIÓN

1. Te ha parecido acertada la música del juego:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. Te han parecido acertados los efectos de sonido del juego:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?
   Todo en general.
2. ¿Qué es lo que menos te ha gustado?
   Que es difícil.
3. ¿Que apartados mejorarías?
   Creo que esta bien segun está.
4. Por favor, indique su valoración general del juego:
   8

Alberto Polo Torres, 25 años.



TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. El juego te ha parecido difícil en general:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

EFECTIVIDAD

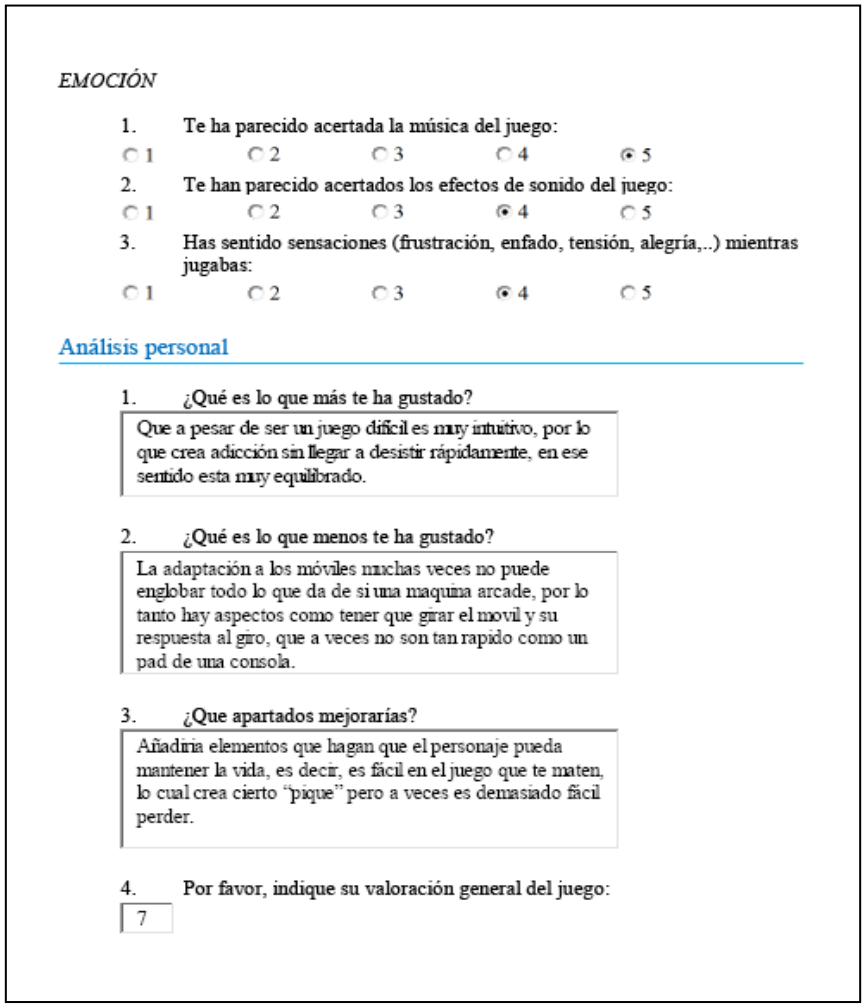
1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. La consideras la interfaz gráfica bien diseñada:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
3. Te parece agradable la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5



EMOCIÓN

1. Te ha parecido acertada la música del juego:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. Te han parecido acertados los efectos de sonido del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?

   Que a pesar de ser un juego difícil es muy intuitivo, por lo que crea adicción sin llegar a desistir rápidamente, en ese sentido esta muy equilibrado.

2. ¿Qué es lo que menos te ha gustado?

   La adaptación a los móviles muchas veces no puede englobar todo lo que da de si una maquina arcade, por lo tanto hay aspectos como tener que girar el movil y su respuesta al giro, que a veces no son tan rapido como un pad de una consola.

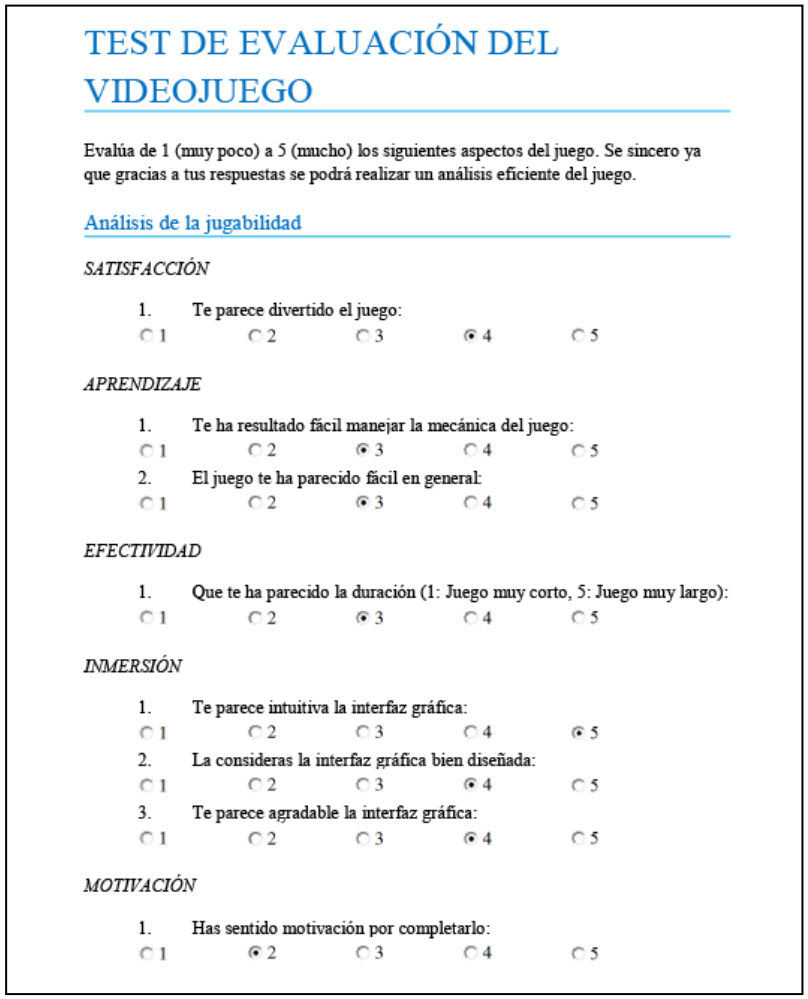
3. ¿Que apartados mejorarías?

   Añadiria elementos que hagan que el personaje pueda mantener la vida, es decir, es fácil en el juego que te maten, lo cual crea cierto "pique" pero a veces es demasiado fácil perder.

4. Por favor, indique su valoración general del juego:

   7

Jonathan Mayoral López-Rey, 25 años.



TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5
2. El juego te ha parecido fácil en general:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

EFECTIVIDAD

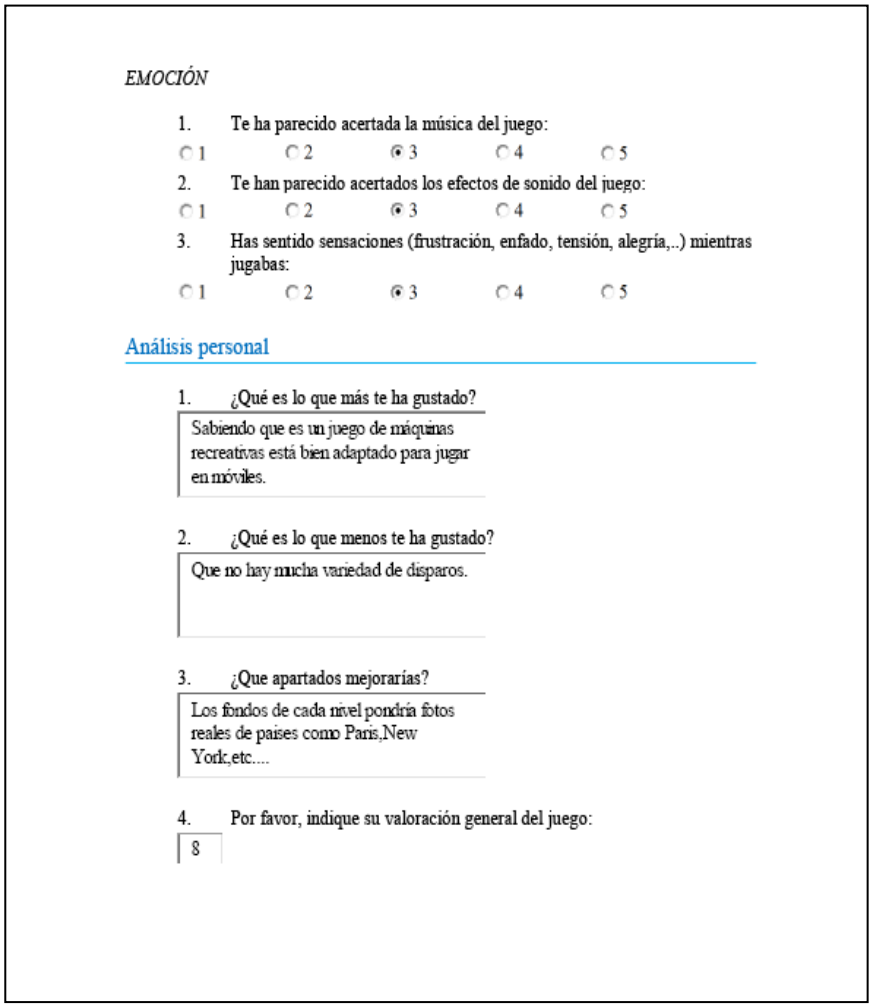
1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. La consideras la interfaz gráfica bien diseñada:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
3. Te parece agradable la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:
   ○ 1 ◉ 2 ○ 3 ○ 4 ○ 5



EMOCIÓN

1. Te ha parecido acertada la música del juego:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5
2. Te han parecido acertados los efectos de sonido del juego:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5
3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?

   Sabiendo que es un juego de máquinas recreativas está bien adaptado para jugar en móviles.

2. ¿Qué es lo que menos te ha gustado?

   Que no hay mucha variedad de disparos.

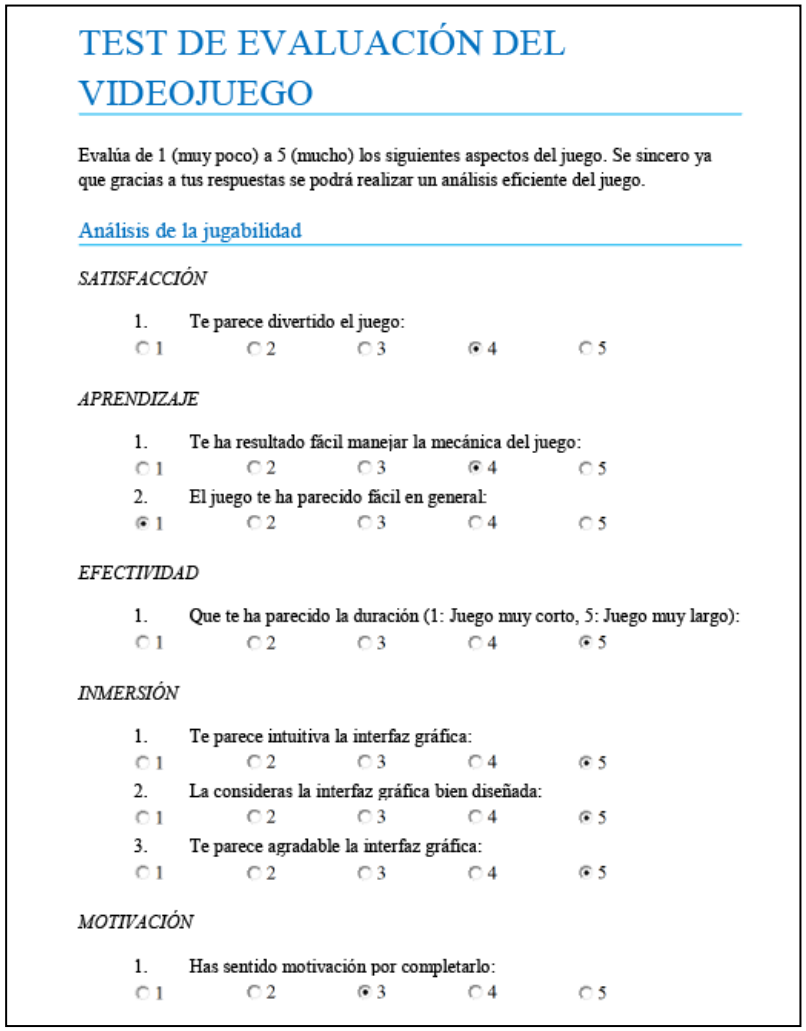
3. ¿Que apartados mejorarías?

   Los fondos de cada nivel pondría fotos reales de paises como Paris,New York,etc....

4. Por favor, indique su valoración general del juego:

   8

Irene Gallardo Manzaneque, 24 años.



TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. El juego te ha parecido fácil en general:
   ◉ 1 ○ 2 ○ 3 ○ 4 ○ 5

EFECTIVIDAD

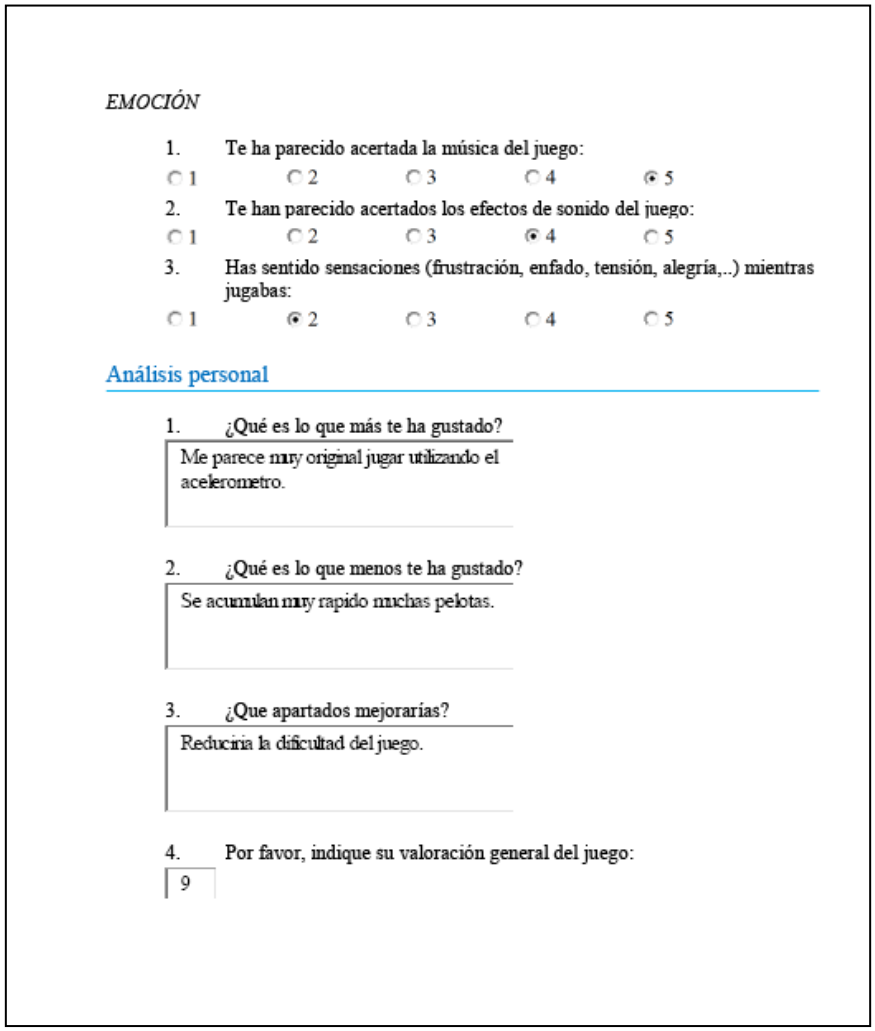
1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. La consideras la interfaz gráfica bien diseñada:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
3. Te parece agradable la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5



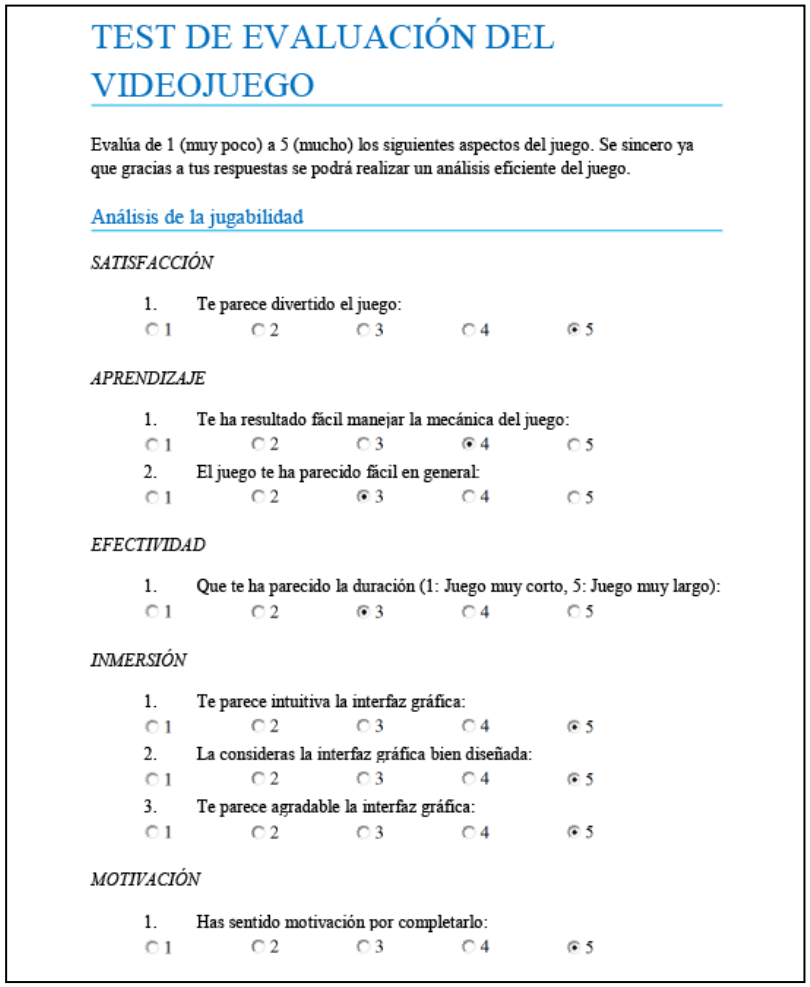
EMOCIÓN

1. Te ha parecido acertada la música del juego:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. Te han parecido acertados los efectos de sonido del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:
   ○ 1 ◉ 2 ○ 3 ○ 4 ○ 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?
   Me parece muy original jugar utilizando el acelerometro.
2. ¿Qué es lo que menos te ha gustado?
   Se acumulan muy rapido muchas pelotas.
3. ¿Que apartados mejorarías?
   Reduciria la dificultad del juego.
4. Por favor, indique su valoración general del juego:
   9

Daniel Aceituno, 24 años.



TEST DE EVALUACIÓN DEL VIDEOJUEGO

Evalúa de 1 (muy poco) a 5 (mucho) los siguientes aspectos del juego. Se sincero ya que gracias a tus respuestas se podrá realizar un análisis eficiente del juego.

Análisis de la jugabilidad

SATISFACCIÓN

1. Te parece divertido el juego:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

APRENDIZAJE

1. Te ha resultado fácil manejar la mecánica del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. El juego te ha parecido fácil en general:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

EFECTIVIDAD

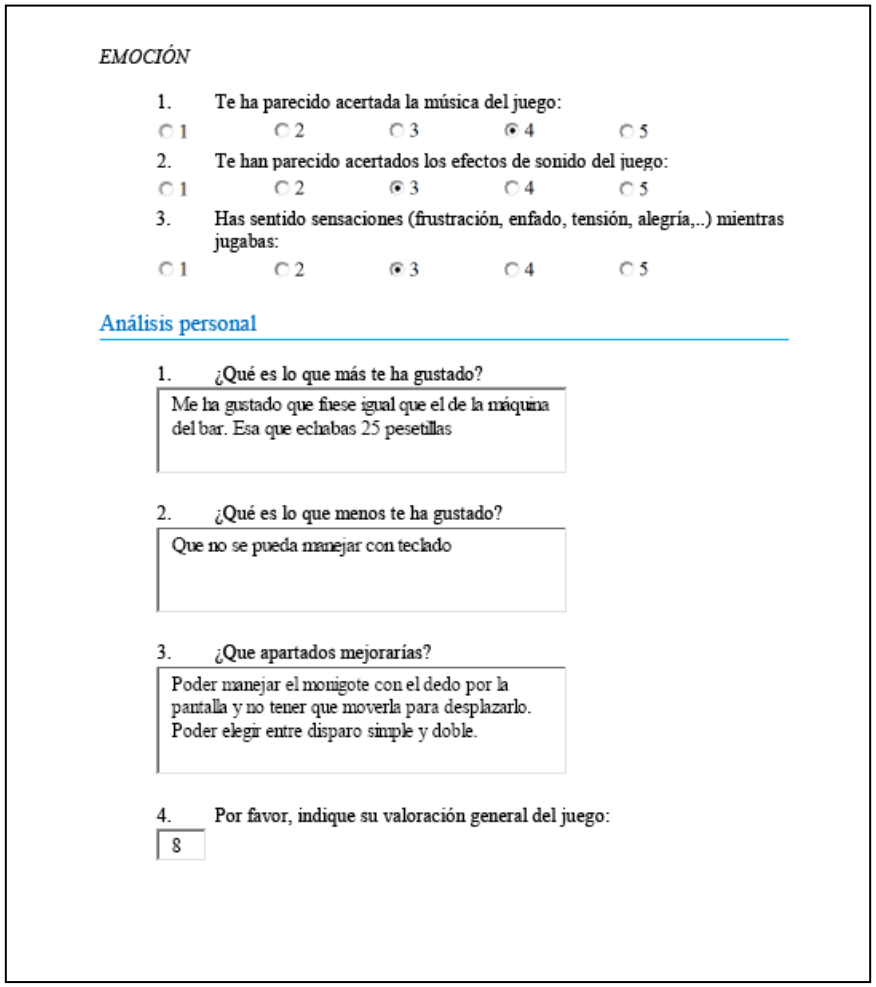
1. Que te ha parecido la duración (1: Juego muy corto, 5: Juego muy largo):
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

INMERSIÓN

1. Te parece intuitiva la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
2. La consideras la interfaz gráfica bien diseñada:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5
3. Te parece agradable la interfaz gráfica:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5

MOTIVACIÓN

1. Has sentido motivación por completarlo:
   ○ 1 ○ 2 ○ 3 ○ 4 ◉ 5



EMOCIÓN

1. Te ha parecido acertada la música del juego:
   ○ 1 ○ 2 ○ 3 ◉ 4 ○ 5
2. Te han parecido acertados los efectos de sonido del juego:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5
3. Has sentido sensaciones (frustración, enfado, tensión, alegría,..) mientras jugabas:
   ○ 1 ○ 2 ◉ 3 ○ 4 ○ 5

Análisis personal

1. ¿Qué es lo que más te ha gustado?
   Me ha gustado que fuese igual que el de la máquina del bar. Esa que echabas 25 pesetillas
2. ¿Qué es lo que menos te ha gustado?
   Que no se pueda manejar con teclado
3. ¿Que apartados mejorarías?
   Poder manejar el monigote con el dedo por la pantalla y no tener que moverla para desplazarlo. Poder elegir entre disparo simple y doble.
4. Por favor, indique su valoración general del juego:
   8

Bibliografía

[1] M. Brownlow. (2011) Smartphone sales and statistics. [Online]. Available: http://www.email-marketing-reports.com/wireless-mobile/smartphone-statistics.htm

[2] Gartner. (2001) Gartner says android to command nearly half of worldwide smartphone operating system market by year-end 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1622614

[3] A. Wells (2011) Android app statistics summary for 2010. [Online]. Available: http://www.androidtapp.com/android-apps-statistics-summary-for-2010/

[4] Research2Guidance. (2011) Android market insights august 2011. [Online]. Available: http://www.research2guidance.com/shop/index.php/downloadable/download/sample/sample_id/148/

[5] J. Rodríguez. (2011) Breve historia de los smartphones. [Online]. Available: http://somosgeeks.wordpress.com/2011/01/17/breve-historia-de-los-smartphones/

[6] Samsung Electronics Co, Ltd. (2011) Samsung's smartphone platform. [Online]. Available: http://www.bada.com/index.html

[7] The Linux Foundation. (2011) An open source, standards-based software platform for multiple devices categories. [Online]. Available: https://www.tizen.org/

[8] Gartner. (2011) Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1764714

[9] teleco.com.br. (2011) Ventas mundiales para usuarios finales y market share. [Online]. Available: http://www.teleco.com.br/es/es_smartphone.asp

[10] phonecurry.com. (2011) The Ultimate Smartphone Shootout. [Online]. Available: http://www.phonecurry.com/knowledge-series/smartphone-comparison/

[11] Danny. (2011) La historia y los comienzos de Android, el sistema operativo de Google. [Online]. Available: http://www.elandroidelibre.com/2011/08/la-historia-y-los-comienzos-de-android-el-sistema-operativo-de-google.html

[12] O. H. Alliance (2011) Open Handset Alliance. [Online]. Available: http://www.openhandsetalliance.com/

[13] Awe. (2011) Historia de la plataforma Android. [Online] Available: http://android.scenebeta.com/tutorial/historia-de-la-plataforma-android

[14] Android developers. (2011) What is Android?. [Online]. Available: http://developer.android.com/guide/basics/what-is-android.html

[15] Android developers (2011) Application Fundamentals. [Online]. Available: http://developer.android.com/guide/topics/fundamentals.html

[16] T. G. Piedrabuena. (2011) Especial: Super Pang. [Online]. Available: http://metodologic.com/index.php?option=com_content&view=article&id=2529:especial-pang&catid=32:metodolog

[17] J. Casanovas. (2004) Usabilidad y arquitectura de software. [Online]. Available: http://www.desarrolloweb.com/articulos/1622.php

[18] sgoliver.net (2010) Estructura de un proyecto Android. [Online]. Available: http://www.sgoliver.net/blog/?p=1278

[19] Android developers (2011) Technical Resources. [Online]. Available: http://developer.android.com/resources/browser.html?tag=tutorial

[20] Android developers (2011) LunarLander. [Online]. Available: http://developer.android.com/resources/samples/LunarLander/index.html

[21] Android developers (2011) XML Layouts. [Online]. Available: http://developer.android.com/guide/topics/ui/declaring-layout.html

[22] tecnoPLOF.com (2010) Jugabilidad. [Online]. Available: http://www.teknoplof.com/2010/09/17/jugabilidad-ese-concepto-que-no-aparece-en-el-diccionario/

[23] J. L. Gonzalez. Sánchez. (2010). Jugabilidad. Caracterización de la experiencia del jugador de videojuegos. [Online]. Available: digibug.ugr.es/bitstream/10481/5671/1/18931200.pdf

[24] Android Market (2011) Aplicaciones. [Online]. Available: https://market.android.com/